City University of Hong Kong

2021 – 2022 Sem B

EE3220 System-on-Chip Design

Lab Report 2

Lab 3: Designing Embedded System with MicroBlaze

Laboratory 4: Creating custom IP for MicroBlaze and PWM

Laboratory 5: Using Seven Segment Display and Pmod with MicroBlaze

Objective of the Experiment

Lab 3:

1. Use Vivado to embody a MicroBlaze soft processor core
2. Add some peripherals to the MicroBlaze
3. Generate bitstream and export hardware
4. Write C program in Vitis IDE
5. Program the MicroBlaze core on the FPGA with Vitis IDE
6. Control the LEDs from the program with Vitis IDE

Lab 4:

1. Create a custom IP by using Vivado
2. Combine the IP with MicroBlaze soft processor
3. Generate pulse width modulation (PWM) signal
4. Use PWM to display various colours on LED

Lab 5:

1. Append third party IP to Vivado
2. Instantiate the PmodSSD in Vivado
3. Combine the IP with MicroBlaze soft processor
4. Use seven-segment multiplexing in MicroBlaze program to display numbers on seven-segment display

Apparatus Used in The Experiment

Hardware:

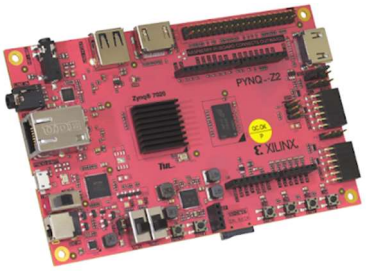


Figure 1: PYNQ-Z2

PYNQ-Z2, is the development board based on Xilinx Zynq SoC xc7z020clg400-1. It provides a Jupyter-based framework with Python APIs for using Xilinx platforms which is suitable for the beginner who design embedded systems without having to use ASIC-style design tools. It has many useful features and interfaces.

Software:

Figure 2: Vivado

Figure 3: Vitis IDE

Vivado, a High-Level Synthesis (HLS) design suite superseding Xilinx ISE, provides an SoC-strength, IP-centric, and system-centric. Vivado transforms a C, C++, or System C into a register transfer level (RTL) that the user can integrate into a Xilinx field programmable gate array (FPGA).

Vitis IDE (Xilinx Vitis), is part of the Vitis unified software platform, designed to develop the embedded software applications targeted toward Xilinx embedded processors. Vitis IDE is based on the Eclipse open-source standard, and it works with hardware designs created by Vivado.

Procedure of The Experiment

Lab3:

There are a total of 2 parts with 4 Checkpoints:

1. Part 1: Embedded system design and Implementation in Vivado

   Create a new project in Vivado with the Pynq-Z2 board, create Block Design of the flow navigator, and add MicroBlaze. Run the Block Automation with changing the memory to 64KB, then regenerate layout and click wizard IP to set the clocks. Click Run connection automation and modify the resets. Add a constant IP with value 1 and connect the resetn and ext_reset_in of the Processor System Reset. Take the snapshot for the block design (Checkpoint 1).

   Add AXI GPIO to the design and rename it to LEDS, select “leds 4 bits” and repeat to add two GPIO IPs with name switches, select “sws 2 bits” and buttons select “btns 4 bits”. Click Run connection automation to regenerate layout. Create a HDL Wrapper by creating a constraint file named “pynq_z2_constr”. Double click the pynq_z2_constr.xdc file and enter the constraints. Generate bitstream and Export Hardware with the bitstream file. Take some snapshots for previous steps (Checkpoint 2).

2. Part 2: Software Development using Vitis

   Create a new Vitis project with the xsa. Write a C file under the src folder to control the LEDs. Build the project and program the FPGA. The LEDs of the development board should be turned on.

   Modify the program to turn on 2 and 3 LEDs separately. Take snapshots. (Checkpoint 3).

   Modify the program to show a binary counting from 1 to 4 (0001 to 0100) on the LEDs. Take snapshots. (Checkpoint 4).

Lab4:

There are a total of 3 parts with 3 Checkpoints:

1. Part 1: Create and Package a custom PWM Controller IP in Vivado

   Create a new project at Vivado and set the Target language to VHDL. Create File “PWM_Controller” with file type VHDL. Create File “pwm_tb”with file type VHDL. Write the test bench code on pwm_tbto verify the PWM controller module. Run simulation

and Run Behavioural Simulation. Then, modify the duty cycles to 20%, 40%, and 60% and observe the change in the pulse widths. (Checkpoint 1)

2. Part2: Build the embedded system with MicroBlaze and the custom PWM IP

   Create a new AXI4 peripheral IP with name pwm_controller, interface with name pwm_AXI. Modify the code, run synthesis and package IP tab. Create a block "MicroBlaze_pwm". Add the MicroBlaze and pwm_controller IP. Then, configure the clock wizard and add a constant. Run connection automation and regenerate layout. Create Port and connect the ports to the outputs of PWM_Controller IP. Create HDL wrapper, generate bitstream and Export Hardware. Take some snapshots of the previous steps. (Checkpoint 2)

3. Part 3: Software Development using Vitis IDE

   Create a new project in Vitis IDE with the xsa file. Write a C file under the src folder. Build the project and program the FPGA. The LEDs of the board should be turned on. Then, modify the program to display pink and yellow colours. (Checkpoint 3)

Lab5:

There are a total of 2 parts with 3 Checkpoints:

1. Part 1: MicroBlaze and Pmod IP Instantiation in Vivado

   Create a new Vivado project. Add the PmodSSD, MicroBlaze, constant, AXI_GPIO as 4-bits buttons. Run connection automation and regenerate layout. Create two new ports named PmodA and PmodB. Regenerate Layout again. Validate the design. Create HDL wrapper, generate bitstream and Export Hardware. Take snapshots of the block design and the bitstream generation. (Checkpoint 1)

2. Part 2: Software Development using Vitis IDE

   Create a new Vitis project with the xsa file. Write a C file under the src folder. Build the project and program the FPGA. The seven-segment display of the board should be displaying numbers. Modify the program to display the last two digits of the SID. (Checkpoint 2) Modify the program to display a count of numbers from 0 to 5. (Checkpoint 3)

Result

Lab3:

1. Check Point 1:

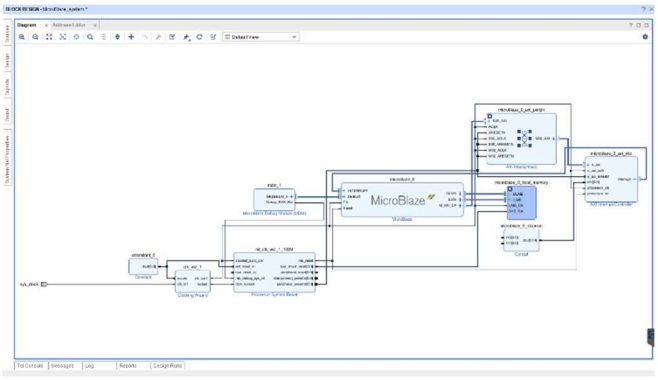


Figure 4: Screenshot of current block design.

2. Check Point 2:

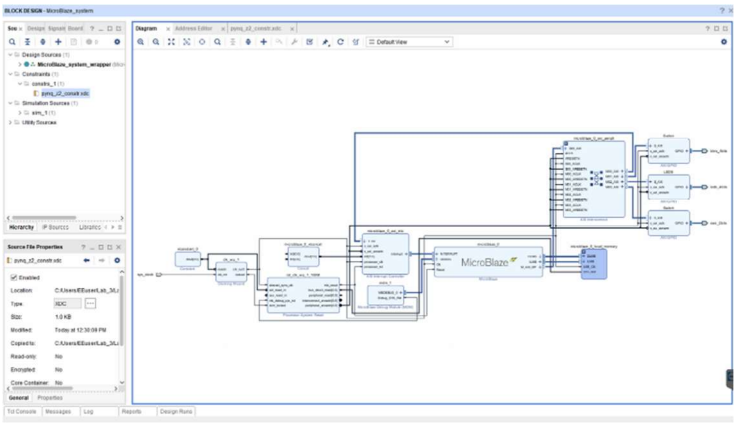


Figure 5: Screenshot of current block design.

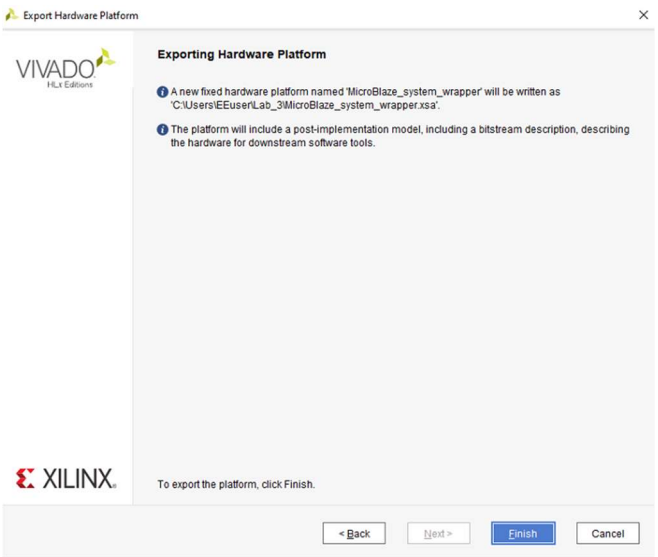


Figure 6: Screenshot of export Hardware Platform.

3. Check Point 3:

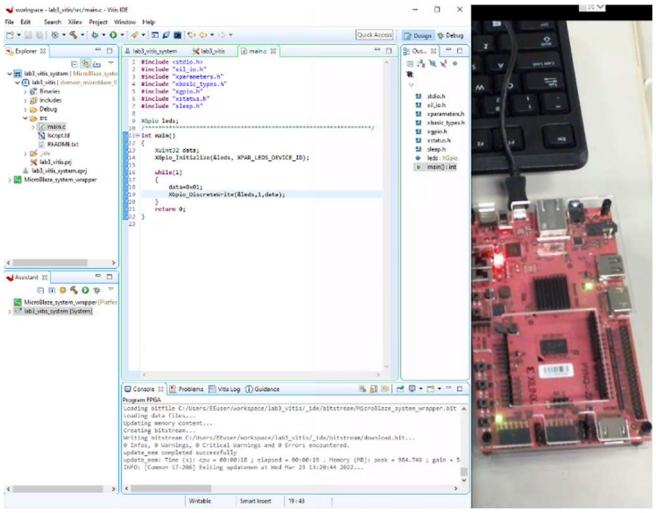


Figure 7: Modified program turning on LED 1.

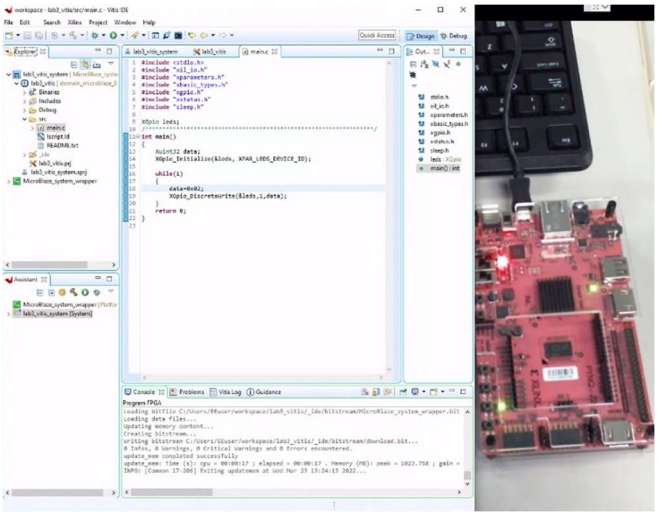


Figure 8: Modified program turning on LED 2.

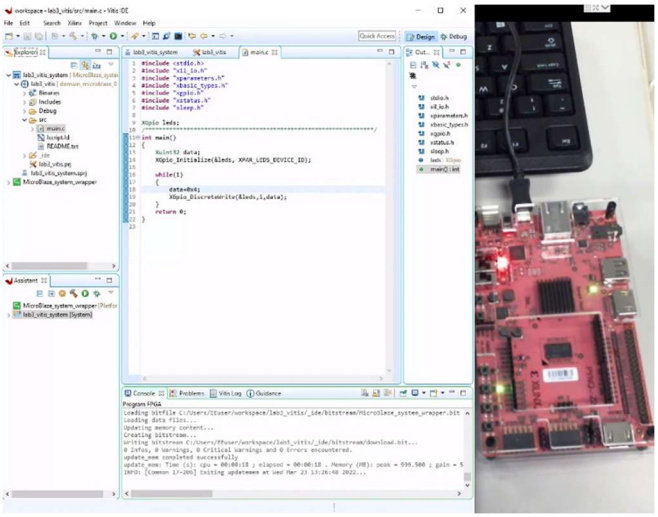


Figure 9: Modified program turning on LED 3.

4. Check Point 4:

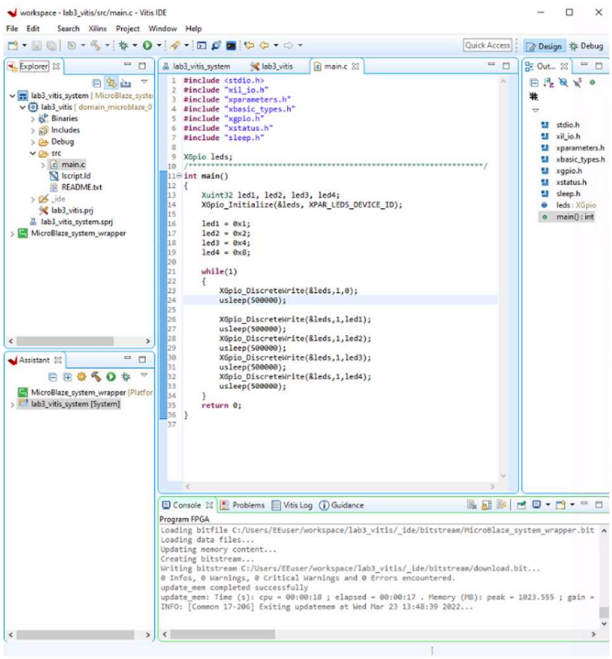


Figure 10: Modified program showing a binary counting on the LEDs.

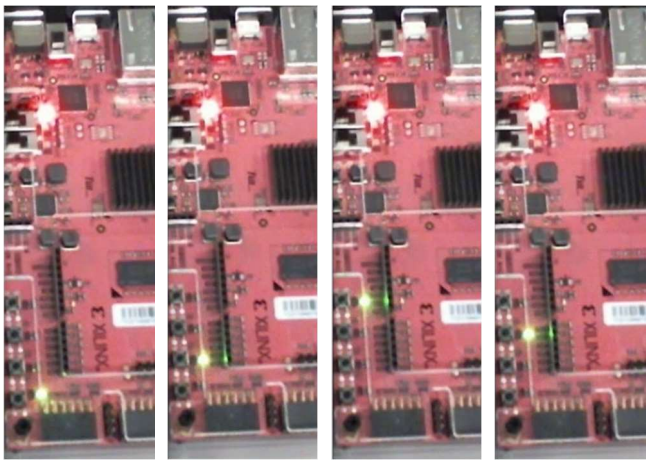


Figure 11: Screenshots of the board showing a binary counting on the LEDs.

Lab4:

1. Check Point 1:

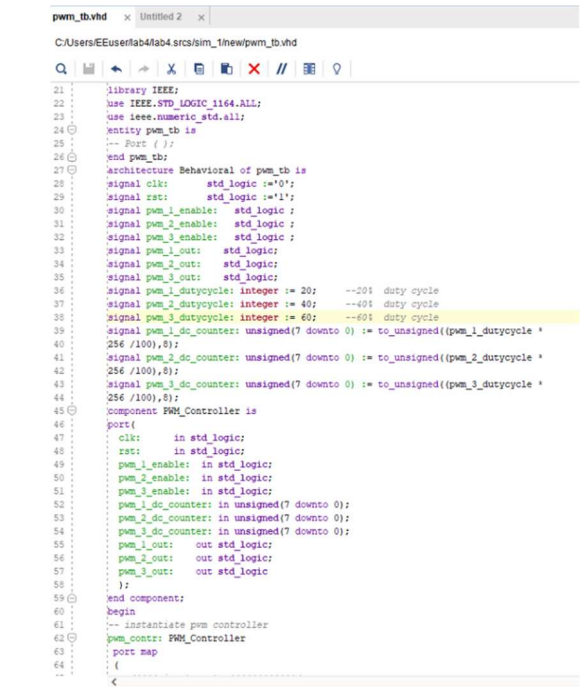


Figure 12: Modified Test Bench Code.

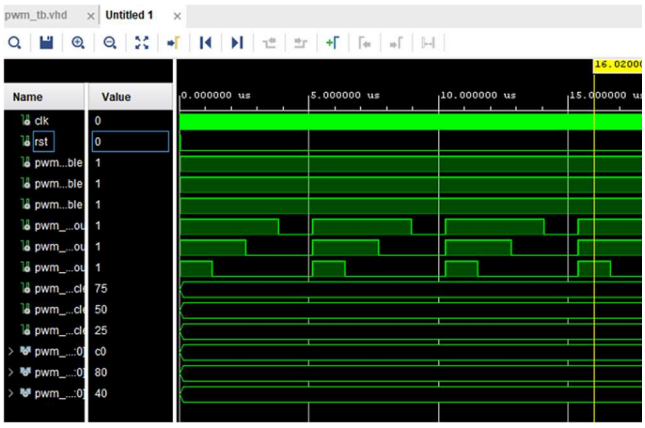


Figure 13: Pulse widths before the changes.

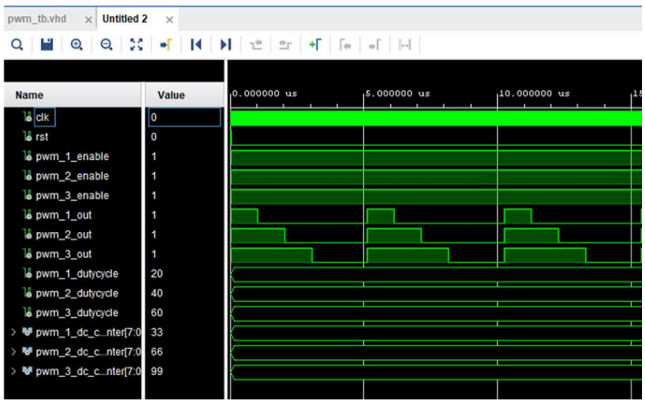


Figure 14: Pulse widths after the changes.

2. Check Point 2:

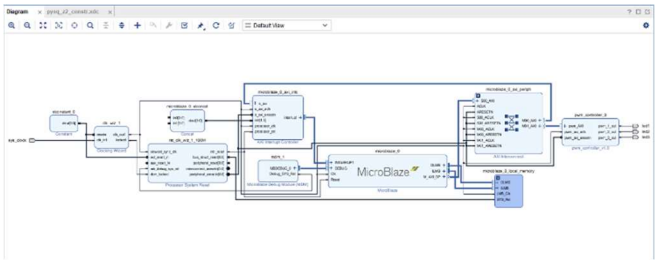


Figure 15: Screenshot of current block design.

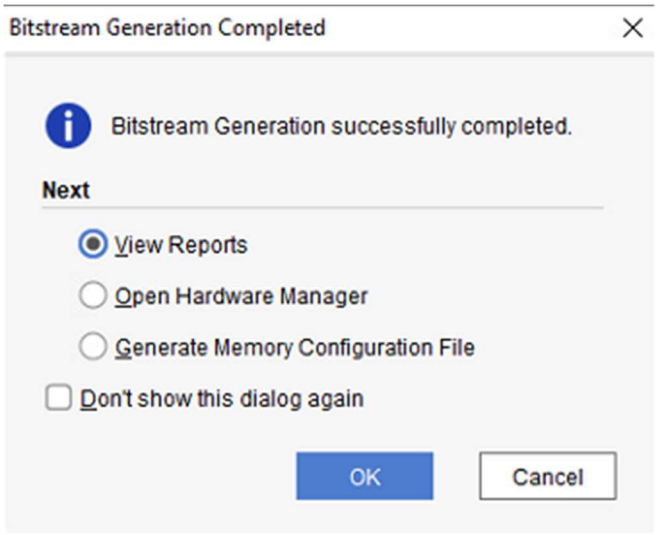


Figure 16: Successfully generated Bitstream.

3. Check Point 3:

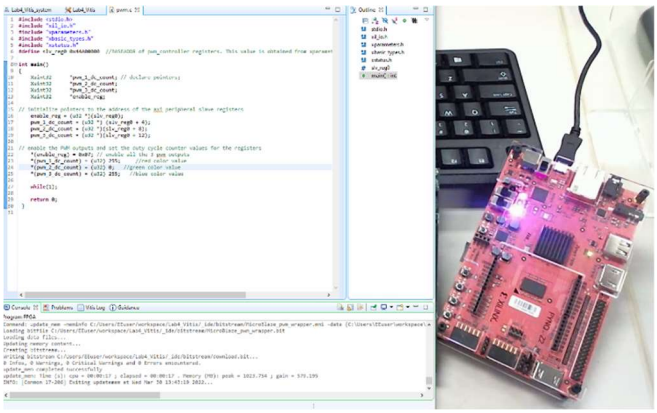


Figure 17: Modified program displaying LED with pink colours.

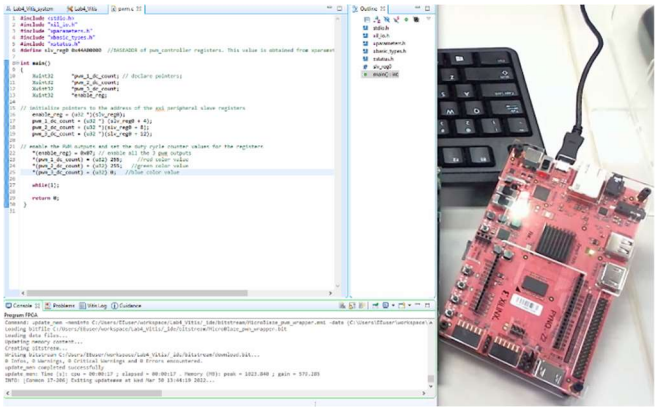


Figure 18: Screenshot of modified program displaying LED with yellow colours.

Lab5:

1. Check Point 1:

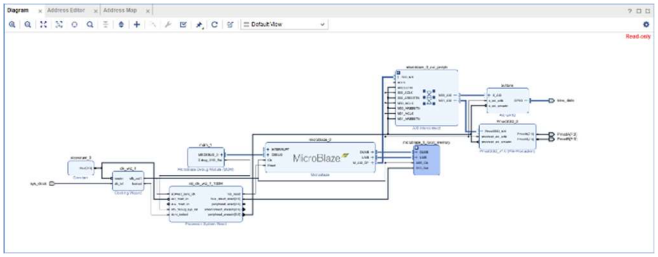


Figure 19: Screenshot of current block design.

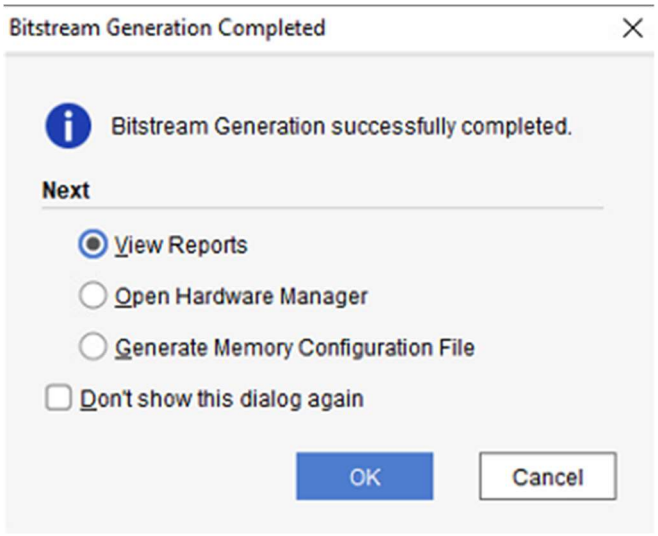


Figure 20: Successfully generated Bitstream File.

2. Check Point 2:

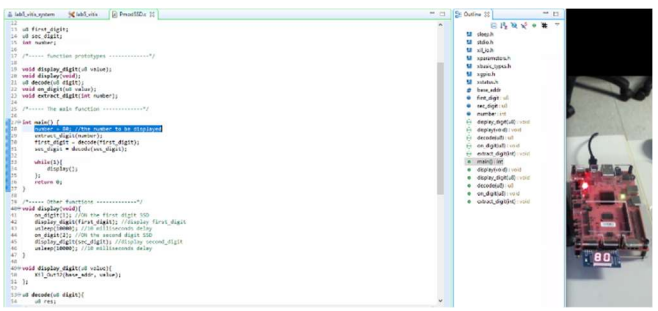


Figure 21: Modified program displaying "80".

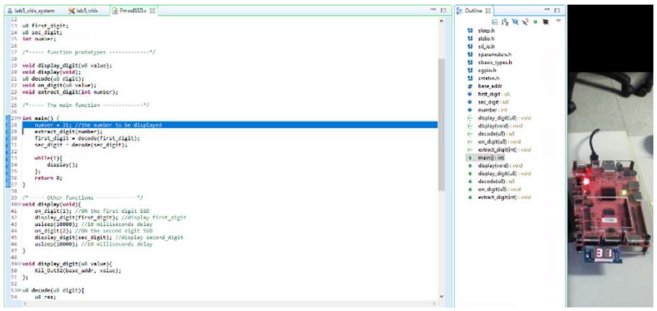


Figure 22: Modified program displaying "31".

3. Check Point 3:

PmodSSD.c

```
/*----- function prototypes ------------*/

void display_digit(u8 value);
void displayNum(int numInput);
void display(void);
u8 decode(u8 digit);
void on_digit(u8 value);
void extract_digit(int number);

/*----- The main function ------------*/

int main() {
    for(int i = 0; i < 7; i++){
        displayNum(i);

        for(int i = 0; i < 20; i++){
            display();
        }
    }

    return 0;
}

void displayNum(int numInput){
    number = numInput; //the number to be displayed
    extract_digit(number);
    first_digit = decode(first_digit);
    sec_digit = decode(sec_digit);
}

/*----- Other functions ------------*/
void display(void){
    on_digit(1); //ON the first digit SSD
    display_digit(first_digit); //display first_digit
    usleep(10000); //10 milliseconds delay
    on_digit(2); //ON the second digit SSD
    display_digit(sec_digit); //display second_digit
    usleep(10000); //10 milliseconds delay
}

void display_digit(u8 value){
    Xil_Out32(base_addr, value);
```

Figure 23: Modified program displaying a count of numbers from 0 to 5.

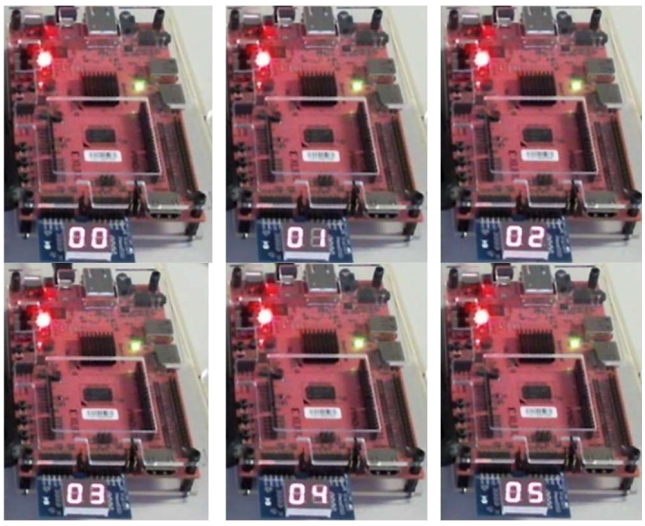


Figure 24: Screenshots of the board displaying a count of numbers from 0 to 5.

Extension work

Modifying program on Lab4 Checkpoint 3 to display LED with RGB colour changing:

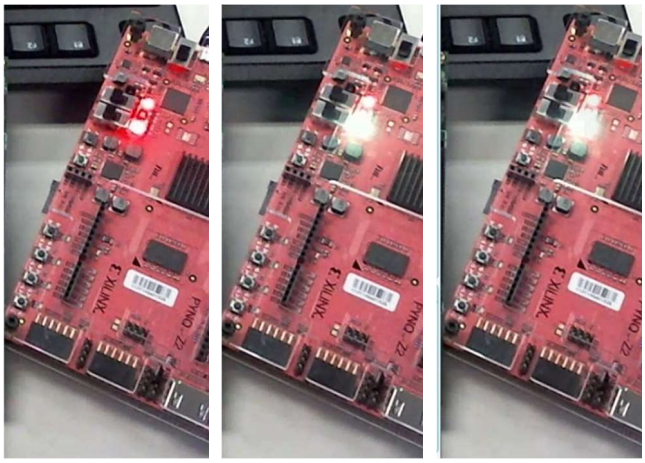
Figure 25: Screenshot of Colour indexing from 1 to 12

```
#define slv_reg0 0x44A00000  //BASEADDR of pwm_controller registers. This value is obtained from xparamet

void pwmRGB(Xuint32 *r, Xuint32 *g, Xuint32 *b, int rVal, int gVal, int bVal, int duration){
    *r = (u32) rVal;    //red color value
    *g = (u32) gVal;   //green color value
    *b = (u32) bVal;   //blue color value
    usleep(duration);

}

int main()
{
    Xuint32      *pwm_1_dc_count; // declare pointers;
    Xuint32      *pwm_2_dc_count;
    Xuint32      *pwm_3_dc_count;
    Xuint32      *enable_reg;

// initialize pointers to the address of the axi peripheral slave registers
    enable_reg = (u32 *)(slv_reg0);
    pwm_1_dc_count = (u32 *) (slv_reg0 + 4);
    pwm_2_dc_count = (u32 *)(slv_reg0 + 8);
    pwm_3_dc_count = (u32 *)(slv_reg0 + 12);

// enable the PWM outputs and set the duty cycle counter values for the registers
    *(enable_reg) = 0x07; // enable all the 3 pwm outputs

    while(1){
        pwmRGB(pwm_1_dc_count, pwm_2_dc_count, pwm_3_dc_count, 255, 0, 0, 200000);
        pwmRGB(pwm_1_dc_count, pwm_2_dc_count, pwm_3_dc_count, 255, 128, 0, 200000);
        pwmRGB(pwm_1_dc_count, pwm_2_dc_count, pwm_3_dc_count, 255, 255, 0, 200000);
        pwmRGB(pwm_1_dc_count, pwm_2_dc_count, pwm_3_dc_count, 128, 255, 0, 200000);
        pwmRGB(pwm_1_dc_count, pwm_2_dc_count, pwm_3_dc_count, 0, 255, 0, 200000);
        pwmRGB(pwm_1_dc_count, pwm_2_dc_count, pwm_3_dc_count, 0, 255, 128, 200000);
        pwmRGB(pwm_1_dc_count, pwm_2_dc_count, pwm_3_dc_count, 0, 255, 255, 200000);
        pwmRGB(pwm_1_dc_count, pwm_2_dc_count, pwm_3_dc_count, 0, 128, 255, 200000);
        pwmRGB(pwm_1_dc_count, pwm_2_dc_count, pwm_3_dc_count, 0, 0, 255, 200000);
        pwmRGB(pwm_1_dc_count, pwm_2_dc_count, pwm_3_dc_count, 127, 0, 255, 200000);
        pwmRGB(pwm_1_dc_count, pwm_2_dc_count, pwm_3_dc_count, 255, 0, 255, 200000);
        pwmRGB(pwm_1_dc_count, pwm_2_dc_count, pwm_3_dc_count, 255, 0, 127, 200000);
    }

    return 0;
}
```

Figure 26: Modified program displaying LED with RGB colour changing.

Color	R	G	B
1	255	0	0
2	255	128	0
3	255	255	0
4	128	255	0
5	0	255	0
6	0	255	128
7	0	255	255
8	0	128	255
9	0	0	255
10	127	0	255
11	255	0	255
12	255	0	127

Table 27: Color with their respective RGB value



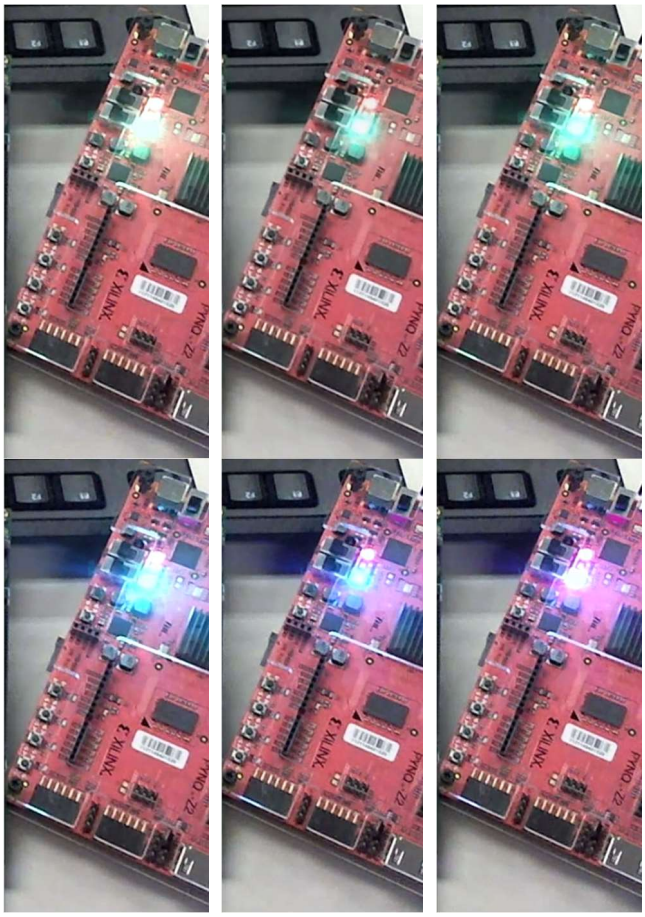


Figure 28: Showcasing of LED with RGB color changing

Analyzed Results

The observed result of the lab is matched with the expected result from the lab instructions to a large extent, in which the behavior of the board, LED, and seven-segment display are matched with what we had written in the code.

In lab 3, as shown in the code, the LEDs 1 to 4 are turned on by setting the value of the function XGpio_DiscreteWrite(). By using this concept, the LEDs 1 to 4 can blink with binary counting by adding a for loop and the usleep() function.

In lab 4, as shown in the code and result, the duty cycle is proportional to PWM output. In addition, the slave register can declare as RGB by PWM signal and output.

In lab 5, as shown in the code, the display number of the seven-segment display by setting the number then passes to various functions. First, pass the number to extract_digit(), it will separate the two digital numbers. Then, call the decode() of each digit to get the SSD value. Lastly, call display() function which will call on_digit() and display_digit() functions to display the digit on the seven-segment display. The on_digit() function will turn on the digits SSD and display_digit() will call the Xil_Out32(), which is provided in xil_io class, to perform an output operation.

Discussion

Inconvenience

As the online mode is adopted for lab sessions, other than the technical difficulties, there are other inconveniences caused by the limitations of adopting online lab sessions. Such as there are only a limited number of TAs in the lab to serve the students which is much higher than the number of TA staff. Thus, when the camera angle and environmental brightness are affected, which damages the video quality and causes difficulties in observing the change of color of the LED and other accessories, which the students have no way to adjust them.

Latency

Since online mode is adopted for lab sessions, the equipment for the lab is connected to the lab computer then to the internet, and the only way for students to control that equipment is by TeamViewer which causes latency on controlling the remote computer, such as editing the C/C++ code on the remote computer or observing the result of the LEDs. Also, the video quality is variable due to the consistence of the connection between the student and the lab is constant.

Conclusion and Summary

Overall, in Lab 3, we learned the basics of how to use Vivado to create and add some peripherals to MicroBlaze IP. Also, we learned how to generate bitstream and export hardware. We got familiar with using Vitis IDE to program the MicroBlaze. Lastly, we grasped how to control the LEDs by the C program.

For Lab 4, we understood how to create the custom IP and generate pulse width modulation (PWM) at Vivado. Furthermore, we recognized how to use PWM to display various colors on RGB LED.

For Lab 5, we learned how to add PmodSSD, which is the third-party IP to Vivado. In addition, we understood how to control the display number on a seven-segment display by modifying the C program.

Reference

[1] Xilinx, "Zynq-7000 SoC Product Selection Guide," 2019. [Online]. Available: https://www.xilinx.com/support/documents/selection-guides/zynq-7000-product-selection-guide.pdf. [Accessed 27 4 2022].

[2] Xilinx, "PYNQ Introduction," 2021. [Online]. Available: https://pynq.readthedocs.io/en/latest/index.html. [Accessed 24 4 2022].

[3] Xilinx, "Vivado," 2022. [Online]. Available: https://www.xilinx.com/support/university/vivado.html. [Accessed 24 4 2022].

[4] Xilinx, "Vitis Unified Software Platform Overview - 2021.2 English," 15 12 2021. [Online]. Available: https://docs.xilinx.com/r/en-US/ug1400-vitis-embedded/Vitis-Unified-Software-Platform-Overview. [Accessed 24 4 2022].

[5] Xilinx, "Vivado Design Suite User High-Level Synthesis," 4 5 2021. [Online]. Available: https://docs.xilinx.com/v/u/en-US/ug902-vivado-high-level-synthesis. [Accessed 24 4 2022].

[6] EE3220: System-On-Chip Design Lab 3: Designing Embedded System with MicroBlaze.

[7] EE3220: System On-Chip Design Lab 4: Creating custom IP for MicroBlaze and PWM.

[8] EE3220: System-On-Chip Design Lab 5: Using Seven Segment Display and Pmod with.