ELC 4396 02
Class Report 4

Jordan Cook

November 7, 2021

1 Introduction and Problem Statement

The purpose of this coding project was to an aesthetically-pleasing night-light. There is a duo of RGB LED's found the Nexys4 DDR board. The goal was to create a pattern where the two lights would cycle through the rainbow, as well as have 4 different levels of brightness which would be controlled by two switches on the board. This would all be accomplished by utilizing pulse-width modulation.

Link to my Vivado Project: doo
Link to the Vitis Project: dink

2 Method and Approach

2.1 Hardware

The first step, as done in the previous report, was to create the FPGA itself. In order to do this, 16 files were downloaded from the textbook author's website (Dr. Pong Chu), as well as the constraint file necessary to program the Nexys board. Those files were:

- chu_gpi.sv
- chu_gpo.sv
- chu_timer.sv
- fifo.sv
- fifo_ctrl.sv
- reg_file.sv
- chu_mmio_controller.sv
- baud_gen.sv
- chu_uart.sv
- uart.sv
- uart_rx.sv
- uart_tx.sv
- chu_io_map.sv
- chu_mcs_bridge.sv
- mmio_sys_vanilla.sv
- mcs_top_vanilla.sv

With these files now included in the project, a CPU from the IP catalog was added to the project to ensure that everything would be working with any system updates.

An additional file was needed to complete this: a pulse-width-modulation file. This was provided by Dr. Pong Chu, which nothing was changed in his System Verilog code. It was connected to the FPGA like usual with other wrappers. In the top level file, an additional set of outputs were

created that would connect the PWM to the two RGB lights. This was everything that was done on the hardware side.

2.2 Software

From this point, an XSA file was exported and opened in the Vitis IDE. After going through the proper sections and creating a platform, certain files from Dr. Chu were needed to be added. These files were:

- chu_init
- chu_init.h
- chu_io_map.h
- chu_io_rw.h
- gpio_cores.cpp
- gpio_cores.h
- timer_core.cpp
- timer_core.h
- uart_core.cpp
- uart_core.h
- main_vanilla_test.cpp

Using classes already defined, a PWM and a GPI variable were created. The PWM would change the output of the colorful LED's and the GPI would provide data on how many of the switches were on (which would also be used as variable in the PWM output);

```
PwmCore myPwm(get_slot_addr(BRIDGE_BASE, S6_PWM));
GpiCore mySwitch(get_slot_addr(BRIDGE_BASE, S3_SW));
```

Listing 1: Core instantiations

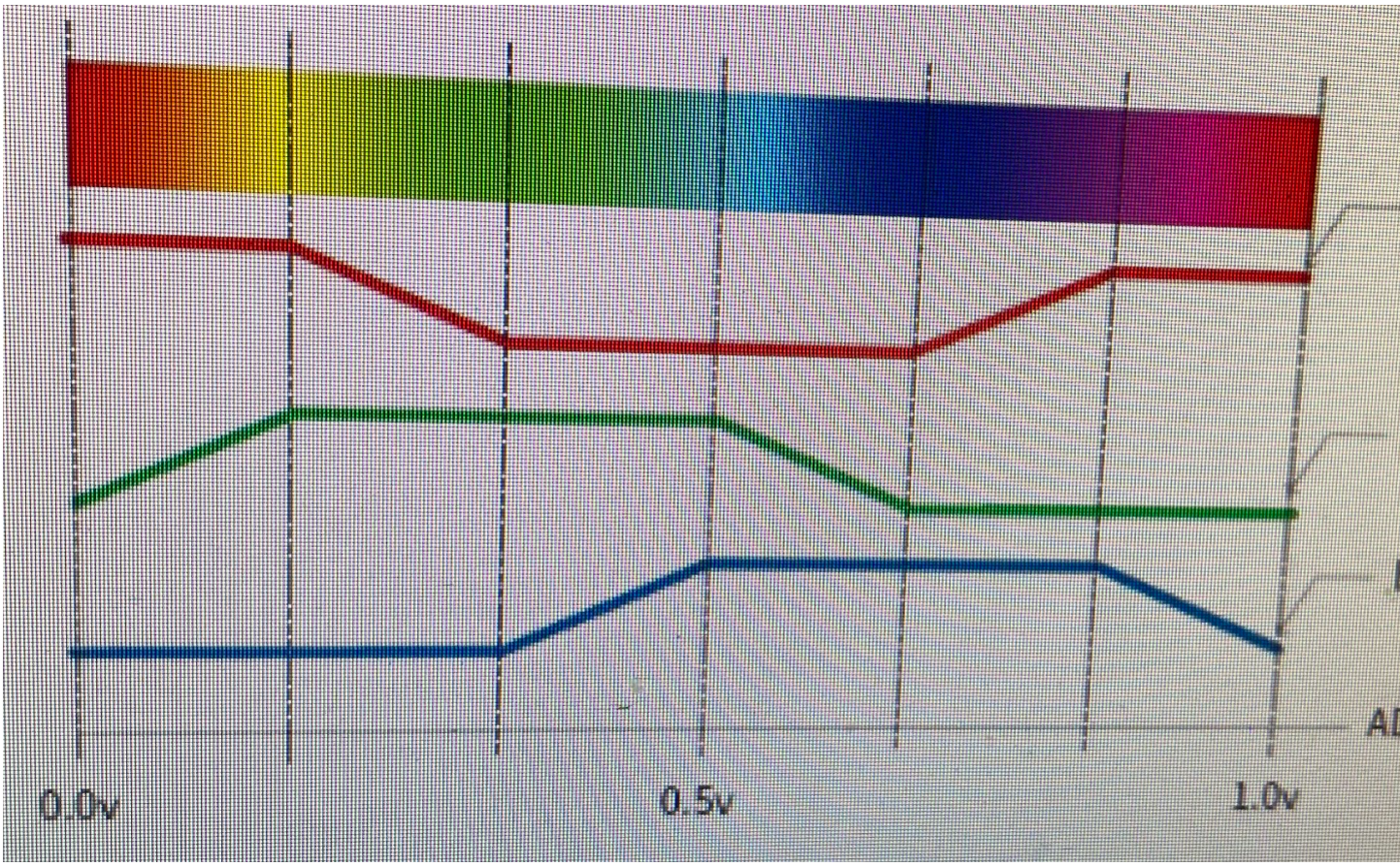
A function was created in the rainbow function to determine which switches were high, which corresponds to how the maximum brightness of the LED's. Then based on which switches are flipped, a specific percentage is assigned that will be a multiplier later on. The original task was to set it in increments of 25%, but for a more obvious change in brightness, it was redesigned to go from 5%, to 30%, to 65%, and finally to full brightness. For debugging, the UART display function was needed to verify that the correct switches were being read (as seen in Putty).

```
double percent;
int switch1;
int switch2;
switch1 = sw_pointer -> read(0);
switch2 = sw_pointer -> read(1);

if (switch1 && switch2) {
  percent = 1.0;
}
else if (!switch1 && switch2) {
  percent = 0.65;
}
else if (switch1 && !switch2) {
  percent = 0.3;
}
else {
  percent = 0.05;
}

uart.disp("Percentage: ");
```

```
21 uart.disp(percent);
22 uart.disp("\n\r");
```

Listing 2: Switch Function

Dr. Chu included some PWM code that would go through each of the three colors, get brighter, and change color. From this code, it was reverse-engineered to get a desired output for the use of this project. The following picture was also a huge aid in figuring out logic that would work:



There were four actions that needed to be done: increase brightness, decrease brightness, set to maximum, or set to zero. This image could also be divided into 6 different sections, where a specific action needs to be done for each of the three LED colors. For example in the first section, red would be set to the maximum, green would be ramping up, and blue would be set to zero. The second section would have red ramping down, green set to maximum, and blue set to zero. This type of pattern would continue for the next 4 segments, where then it would simply repeat again and be back at square-one. The code for the whole function used is shown below:

```
1 void rainbowLED(PwmCore *pwm, GpiCore *sw_pointer) {
2
3   double bright = 1.0;
4   double duty;
5   double P = 1.2589;
6
7   const int length = 6 * 20;
8
9   int red = 0;
```

```
10  int green = 1;
11  int blue = 2;
12
13  pwm -> set_freq(50);
14
15  for (int i = 0; i < length; i++) {
16
17    double percent;
18    int switch1;
19    int switch2;
20    switch1 = sw_pointer -> read(0);
21    switch2 = sw_pointer -> read(1);
22
23    if (switch1 && switch2) {
24      percent = 1.0;
25    }
26    else if (!switch1 && switch2) {
27      percent = 0.65;
28    }
29    else if (switch1 && !switch2) {
30      percent = 0.3;
31    }
32    else {
33      percent = 0.05;
34    }
35
36    if (i <=20) {
37      bright = bright * P;
38      duty = bright / 100.0;
39
40      pwm -> set_duty(1.0 * percent, red);
41      pwm -> set_duty(1.0 * percent, red + 3);
42
43      pwm -> set_duty(duty * percent, green);
44      pwm -> set_duty(duty * percent, green + 3);
45
46      pwm -> set_duty(0, blue);
47      pwm -> set_duty(0, blue + 3);
48
49      sleep_ms(100);
50    }
51    else if (i > 20 && i <= 40) {
52      bright = bright / P;
53      duty = bright / 100.0;
54
55      pwm -> set_duty(duty * percent, red);
56      pwm -> set_duty(duty * percent, red + 3);
57
58      pwm -> set_duty(1.0 * percent, green);
59      pwm -> set_duty(1.0 * percent, green + 3);
60
61      pwm -> set_duty(0, blue);
62      pwm -> set_duty(0, blue + 3);
63
64      sleep_ms(100);
65    }
66    else if (i > 40 && i <= 60) {
67      bright = bright * P;
68      duty = bright / 100.0;
```

```
69
70      pwm -> set_duty(0, red);
71      pwm -> set_duty(0, red + 3);
72
73      pwm -> set_duty(1.0 * percent, green);
74      pwm -> set_duty(1.0 * percent, green + 3);
75
76      pwm -> set_duty(duty * percent, blue);
77      pwm -> set_duty(duty * percent, blue + 3);
78
79      sleep_ms(100);
80    }
81    else if (i > 60 && i <= 80) {
82      bright = bright / P;
83      duty = bright / 100.0;
84
85      pwm -> set_duty(0, red);
86      pwm -> set_duty(0, red + 3);
87
88      pwm -> set_duty(duty * percent, green);
89      pwm -> set_duty(duty * percent, green + 3);
90
91      pwm -> set_duty(1.0 * percent, blue);
92      pwm -> set_duty(1.0 * percent, blue + 3);
93
94      sleep_ms(100);
95    }
96    else if (i > 80 && i <= 100) {
97      bright = bright * P;
98      duty = bright / 100.0;
99
100     pwm -> set_duty(duty * percent, red);
101     pwm -> set_duty(duty * percent, red + 3);
102
103     pwm -> set_duty(0, green);
104     pwm -> set_duty(0, green + 3);
105
106     pwm -> set_duty(1.0 * percent, blue);
107     pwm -> set_duty(1.0 * percent, blue + 3);
108
109     sleep_ms(100);
110   }
111   else if (i > 100 && i <= 120) {
112     bright = bright / P;
113     duty = bright / 100.0;
114
115     pwm -> set_duty(1.0 * percent, red);
116     pwm -> set_duty(1.0 * percent, red + 3);
117
118     pwm -> set_duty(0, green);
119     pwm -> set_duty(0, green + 3);
120
121     pwm -> set_duty(duty * percent, blue);
122     pwm -> set_duty(duty * percent, blue + 3);
123
124     sleep_ms(100);
125   }
126 }
```

```
127 }
```

Listing 3: Rainbow LED Function

The final step was to call this function and only this one from main. A PWM pointer and a GPI pointer were the arguments from the function.

```
1 int main() {
2   while (1) {
3     rainbowLED(&pwm, &mySwitch);
4   }
5 }
```

Listing 4: Main

3 Tests

There were many checkpoints and steps gone through to verify the functionality of this program. The list as completed goes:

1. Add hardware and see if Bitstream can be generated. Once successful, export the hardware.
2. Implement the base code for a PWM and see if the RGB's respond as expected.
3. Get the lights to operate in a simple rainbow pattern with a constant brightness.
4. Read in the switch values. Verify their values using UART.
5. Adjust the brightness by a factor based on switches.

Each of these steps were tested to see if they had been completed properly. This ensured a system without errors and expedited the debugging process.

4 Results

The finished product was two RGB lights that went in a cycle of changing rainbow colors. The first two switches on the board changed their brightness

5 Conclusion

The project demonstrated the functionality of a PWM output. Since digital systems cannot changed the value of their output, PWM allows a pseudo-analog output by providing an average at a rate that the human eye cannot detect and therefore perceives a smooth transition. This project also showed a neat light-trick by simply changing the values of 3 colors, but is perceived to be a smooth rainbow pattern. All in all, this was a fun and interesting learning experience.