Ram Bhattarai
ECE 544
Project 1
Roy Kravitz
April 19th 2019

Project Description:

The main goal of this project is to give us an understanding of how to create an embedded system application using Xilinx Vivado and SDK embedded system tools. The project adds two functionalities ,pulse width modulation and pulse width detection, to the existing getting started project. The getting started project builds a target hardware platform for the project and a sample application to provide a basis for project 1. The getting started project came with the custom peripheral and drivers for Nexys4IO. The custom peripheral drives the pushbuttons, switches, and LEDs on the Digilent Nexys A7 board. The project adds two additional peripherals PMODENC to add a rotary encoder and PMODOLEDrgb to provide the color display output. These additional devices are controlled by two different pieces of IP, the PmodOLEDrgb_V1.0 and PmodEnc_v1.0.

Functional Specification:

In project 1, we will generate a variable duty-cycle PWM output signal for the on board RGB LED's and will detect the duty cycle using both hardware and software pulse width detection. The generation of the PWM for the RGB LED's is done by implementing a color wheel application. A color wheel lets a user choose any color by varying the Hue, Saturation and the Value in the HSV scale. We implement the hardware pulse width detection using the RGB LEDS on the board. For this project, I added three gpios for calculating the RED, GREEN, and BLUE high and low count values.

Peripherals used in the project:

- ❏ Slide Switch [0] - Selects between hardware and software pulse width detection. Hardware pulse width detection should be enabled when the switch is on(up). Software pulse width detection should be enabled when the switch is down(off).
- ❏ Rotary Encoder - The rotary encoder knob on the PmodENc is used to alter the Hue value of the color wheel. As hue changes, the color values change as well. Pressing the push button on the encoder will terminate the application.
- ❏ Pushbuttons - System reset is done by pressing the CPU Reset button(Red button on the board). The button is mapped to the sysreset_n(asserted low) and sysreset(asserted high) signals in the top level
- ❏ PmodOLEDrgb display - The display should show the Hue, Saturation, and Value on the left side and show the corresponding color value on in a rectangle box.
- ❏ LEDs on the Nexys A7 board:
    - ❏ LED[0]- provides the visual indication of pwm detection. If lit, it should do hardware detection, else do the detection from the software

    - ❏ RGB1 & RGB2 - Blink the LEDs with the value coming from the color wheel application. It is the same color displayed on the PmodOLEDrgb.

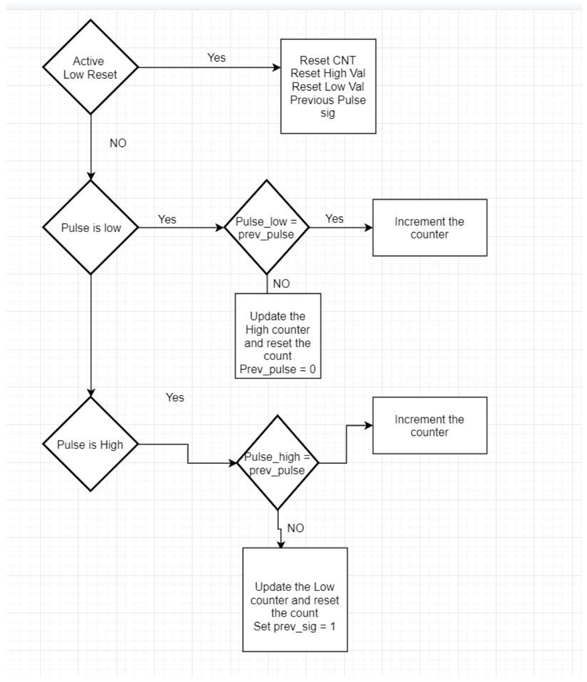
- ❏ Seven Segment Display-
    - ❏ Digits [7:6] - Display the RED duty cycle for the RGB Led
    - ❏ Digits [4:3] - Display the GREEN duty cycle for the RGB Led
    - ❏ Digits [1:0] - Display the BLUE duty cycle for the RGB Led
    - ❏ Digits [2] and Digits[5] should be left blank

Implementation:

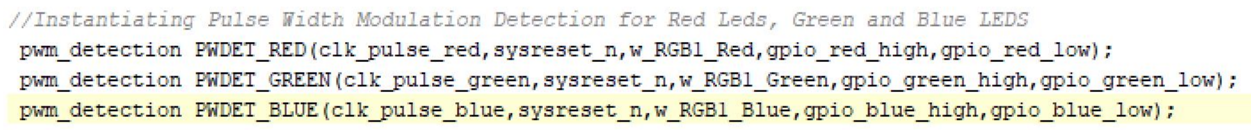
Hardware:

Implement the pulse width detection on Hardware using the system verilog subset of verilog. The pulse width detection is done in hardware by counting the high and low times of the signal. Each time when the pulse is low or high the counter is incremented. Flowchart below shows the implementation

Flowchart for the algorithm:

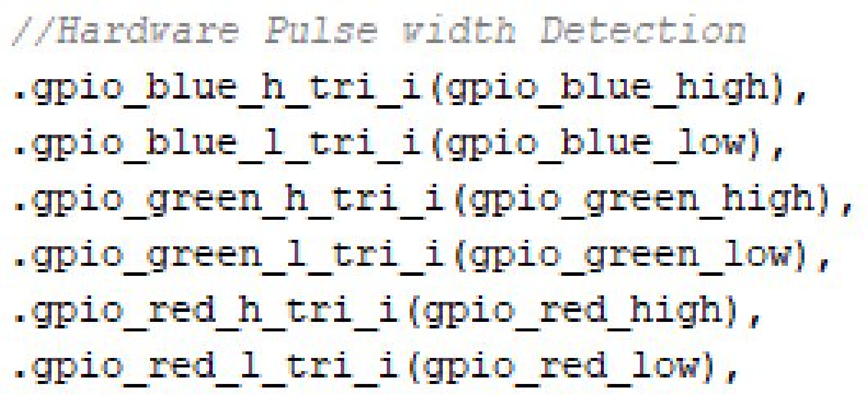

Active Low Reset
Yes
Reset CNT Reset High Val Reset Low Val Previous Pulse sig
NO
Pulse is low
Yes
Pulse_low = prev_pulse
Yes
Increment the counter
NO
Update the High counter and reset the count Prev_pulse = 0
Yes
Pulse is High
Pulse_high = prev_pulse
Increment the counter
NO
Update the Low counter and reset the count Set prev_sig = 1

The module created is instantiated in top module 3 times. Each instantiation is for detecting pulse for RED, GREEN and BLUE. Instantiation is shown in a code snippet below:



```
//Instantiating Pulse Width Modulation Detection for Red Leds, Green and Blue LEDS
 pwm_detection PWDET_RED(clk_pulse_red,sysreset_n,w_RGB1_Red,gpio_red_high,gpio_red_low);
 pwm_detection PWDET_GREEN(clk_pulse_green,sysreset_n,w_RGB1_Green,gpio_green_high,gpio_green_low);
 pwm_detection PWDET_BLUE(clk_pulse_blue,sysreset_n,w_RGB1_Blue,gpio_blue_high,gpio_blue_low);
```

gpio_red_high, gpio_red_low, gpio_green_high, gpio_green_low, gpio_blue_high, gpio_blue_low are the signals connected to the embedded systems. 3 AXI_GPIOs are added to the embedded system.



```
//Hardware Pulse width Detection
.gpio_blue_h_tri_i(gpio_blue_high),
.gpio_blue_l_tri_i(gpio_blue_low),
.gpio_green_h_tri_i(gpio_green_high),
.gpio_green_l_tri_i(gpio_green_low),
.gpio_red_h_tri_i(gpio_red_high),
.gpio_red_l_tri_i(gpio_red_low),
```

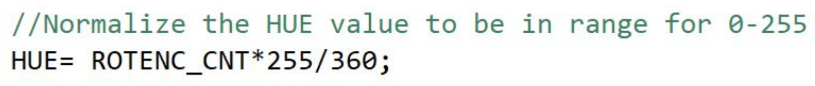
A slow clock is also added to the system because sysclk in the system operates at a higher frequency and we need to slow down. The slowest clock that can be generated is 5 Mhz.

Software:

Color Wheel application: The color wheel is implemented by taking the switches and rotary encoder input and displaying the color according to the HSV scale.

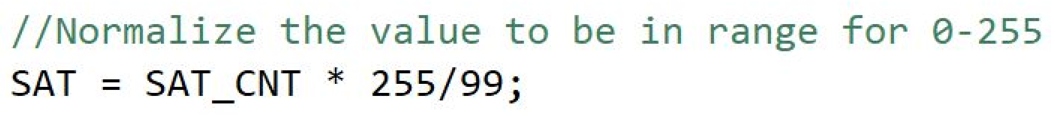
Steps:

1) Initialize the laststate
2) Poll in a while loop
    a) Initialize the state
    b) Check if the rotary encoder push button or the center push button is pressed, if yes, close the application
    c) Keep reading the encoder values
    d) Check if the encoder reaches 359, if it is greater than 359 reset it back to 0
    e) Check if the encoder values is less than 0, reset it to 359
    f) Normalize the encoder value to be in range of 0-255



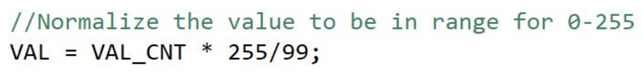
```
//Normalize the HUE value to be in range for 0-255
HUE= ROTENC_CNT*255/360;
```

g) Check if the Right button is pressed and the value is less than 99, yes increment the saturation count
h) If the value reaches to 100 reset it back to 0
i) Check if the Left push button is pressed and the value is is greater than 0, yes decrement, no set it back to 99
j) Normalize the saturation counter value to be in a range



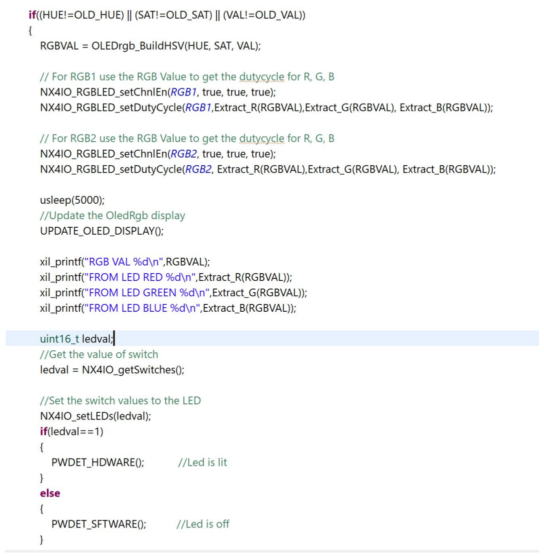
```
//Normalize the value to be in range for 0-255
SAT = SAT_CNT * 255/99;
```

k) Check if the UP push button is pressed and the value is less than 99, yes increment the value count
l) If the value reaches to 100 reset it back to 0
m) Check if the Down push button is pressed and the value is is greater than 0, yes decrement, no set it back to 99
n) Normalize the Value counter value to be in a range



```
//Normalize the value to be in range for 0-255
VAL = VAL_CNT * 255/99;
```

o)



```
if((HUE!=OLD_HUE) || (SAT!=OLD_SAT) || (VAL!=OLD_VAL))
{
   RGBVAL = OLEDrgb_BuildHSV(HUE, SAT, VAL);

   // For RGB1 use the RGB Value to get the dutycycle for R, G, B
   NX4IO_RGBLED_setChnlEn(RGB1, true, true, true);
   NX4IO_RGBLED_setDutyCycle(RGB1,Extract_R(RGBVAL),Extract_G(RGBVAL), Extract_B(RGBVAL));

   // For RGB2 use the RGB Value to get the dutycycle for R, G, B
   NX4IO_RGBLED_setChnlEn(RGB2, true, true, true);
   NX4IO_RGBLED_setDutyCycle(RGB2, Extract_R(RGBVAL),Extract_G(RGBVAL), Extract_B(RGBVAL));

   usleep(5000);
   //Update the OledRgb display
   UPDATE_OLED_DISPLAY();

   xil_printf("RGB VAL %d\n",RGBVAL);
   xil_printf("FROM LED RED %d\n",Extract_R(RGBVAL));
   xil_printf("FROM LED GREEN %d\n",Extract_G(RGBVAL));
   xil_printf("FROM LED BLUE %d\n",Extract_B(RGBVAL));

   uint16_t ledval;
   //Get the value of switch
   ledval = NX4IO_getSwitches();

   //Set the switch values to the LED
   NX4IO_setLEDs(ledval);
   if(ledval==1)
   {
      PWDET_HDWARE();        //Led is lit
   }
   else
   {
      PWDET_SFTWARE();       //Led is off
   }
```

p) Initialize the GPIO registers in the INIT_MODULE

```
    ,
// GPIO0 channel 1 is an 8-bit input port.
// GPIO0 channel 2 is an 8-bit output port.
XGpio_SetDataDirection(&GPIOInst0, GPIO_0_INPUT_0_CHANNEL, 0xFF);
XGpio_SetDataDirection(&GPIOInst0, GPIO_0_OUTPUT_0_CHANNEL, 0x00);

// GPIO1 channel 1 is an 32-bit input port
// GPIO1 channel 2 is an 32-bit input port
// Used to detect the High and Low values for RED Leds of RGB LED
XGpio_SetDataDirection(&GPIOInst1, GPIO_1_INPUT_0_CHANNEL, 0xFFFFFFFF);
XGpio_SetDataDirection(&GPIOInst1, GPIO_1_INPUT_1_CHANNEL, 0xFFFFFFFF);

// GPIO2 channel 1 is an 32-bit input port
// GPIO2 channel 2 is an 32-bit input port
// Used to detect the High and Low values for GREEN Leds of RGB LED
XGpio_SetDataDirection(&GPIOInst2, GPIO_2_INPUT_0_CHANNEL, 0xFFFFFFFF);
XGpio_SetDataDirection(&GPIOInst2, GPIO_2_INPUT_1_CHANNEL, 0xFFFFFFFF);


// GPIO2 channel 1 is an 32-bit input port
// GPIO2 channel 2 is an 32-bit input port
// Used to detect the High and Low values for BLUE Leds of RGB LED
XGpio_SetDataDirection(&GPIOInst3, GPIO_3_INPUT_0_CHANNEL, 0xFFFFFFFF);
XGpio_SetDataDirection(&GPIOInst3, GPIO_3_INPUT_1_CHANNEL, 0xFFFFFFFF);
```

q) Pulse width is detected in software using the same algorithm as hardware
We will monitor for the low and high values for the signals in the fit timer module.
Gpio_in needs to be shifted by the right amount to get to the RGB values.

```
// Read the GPIO port to read back the generated PWM signal for RGB led's
gpio_in = XGpio_DiscreteRead(&GPIOInst0, GPIO_0_INPUT_0_CHANNEL);


SW_RED = (gpio_in & 0x0004)>>2;                 //Shift to get to the SW_RED_register
SW_BLUE = (gpio_in & 0x0002)>>1;                //Shift to get to the SW_Blue Register
SW_GREEN =(gpio_in & 0x0001)>>0;                //Shift to get to the SW_BLUE register
```

r) Hardware Detection is done by reading the GPIO registers that we initially implemented in the hardware steps:

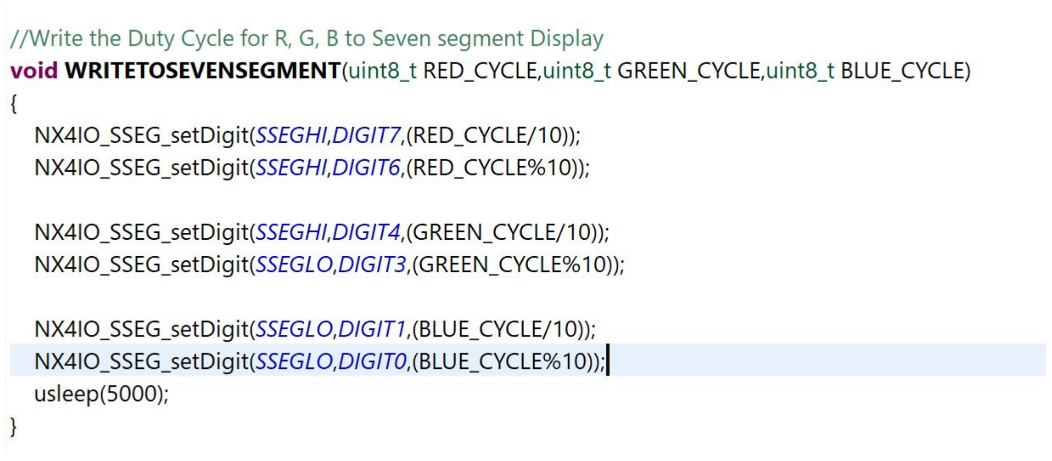
```
PWDET_RED_H = Gpio_DiscreteRead(&GPIOInst1,GPIO_1_INPUT_0_CHANNEL);
PWDET_RED_L = Gpio_DiscreteRead(&GPIOInst1,GPIO_1_INPUT_1_CHANNEL);

PWDET_GREEN_H = Gpio_DiscreteRead(&GPIOInst2,GPIO_2_INPUT_0_CHANNEL);
PWDET_GREEN_L = Gpio_DiscreteRead(&GPIOInst2,GPIO_2_INPUT_1_CHANNEL);

PWDET_BLUE_H = Gpio_DiscreteRead(&GPIOInst3,GPIO_3_INPUT_0_CHANNEL);
PWDET_BLUE_L = Gpio_DiscreteRead(&GPIOInst3,GPIO_3_INPUT_1_CHANNEL);
```

s) Duty cycle calculation
(Pulse High * 100)/(Pulse High + Pulse Low)

t) Seven Segment Display:



```
//Write the Duty Cycle for R, G, B to Seven segment Display
void WRITETOSEVENSEGMENT(uint8_t RED_CYCLE,uint8_t GREEN_CYCLE,uint8_t BLUE_CYCLE)
{
   NX4IO_SSEG_setDigit(SSEGHI,DIGIT7,(RED_CYCLE/10));
   NX4IO_SSEG_setDigit(SSEGHI,DIGIT6,(RED_CYCLE%10));

   NX4IO_SSEG_setDigit(SSEGHI,DIGIT4,(GREEN_CYCLE/10));
   NX4IO_SSEG_setDigit(SSEGLO,DIGIT3,(GREEN_CYCLE%10));

   NX4IO_SSEG_setDigit(SSEGLO,DIGIT1,(BLUE_CYCLE/10));
   NX4IO_SSEG_setDigit(SSEGLO,DIGIT0,(BLUE_CYCLE%10));
   usleep(5000);
}
```

References:

- https://en.wikipedia.org/wiki/HSL_and_HSV
- https://stackoverflow.com/questions/3018313/algorithm-to-convert-rgb-to-hsv-and-hsv-to-rgb-in-range-0-255-for-both
- Digilent Nexys A7 Board Reference Manual. Copyright Digilent, Inc.
- Digilent PmodOLEDrgb Reference Manual. Copyright Digilent, Inc.
- Digilent PmodENC Reference Manual. Copyright Digilent, Inc.
- Getting Started in ECE 544(Vivado/Nexys A7) by Roy Kravitz
- https://www.lifewire.com/what-is-hsv-in-design-1078068