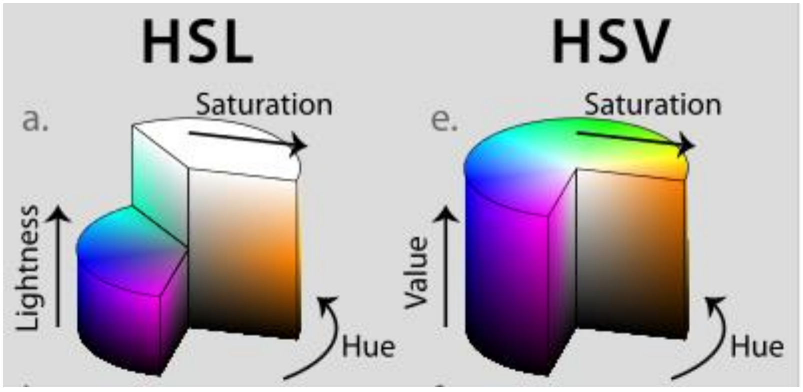
Name: Hai Dang Hoang

Email: danghai@pdx.edu

Design Report project 1

1. Introduction:

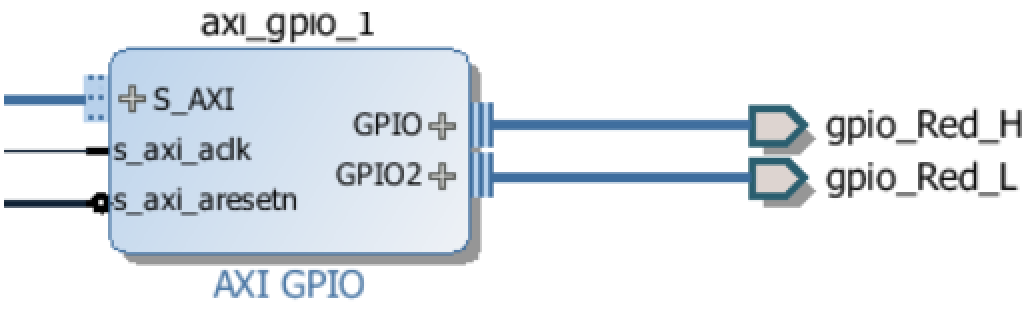
Generate a variable duty-cycle PWM output signal for the on board RGB LED's and you will detect the duty cycle of that PWM output by hardware you design and by implementing a pulse-width detection algorithm in application software. The generation of the PWM for the RGB LED's is done by the implementation of a Colorwheel. A Colorwheel is something from which you can choose any color by varying the Hue, Saturation and the Value in the HSV scale.



(Reference: https://en.wikipedia.org/wiki/HSL_and_HSV)

2. Additional hardware inserted:

By completing the tasks in "Getting Started in ECE544" (Vivado/SDK), I can create the Microblaze-based embedded system. Additionally , 3 more axi_gpio components are inserted to the block design



Configuration:

+ GPIO Width 32 bits, all inputs is checked

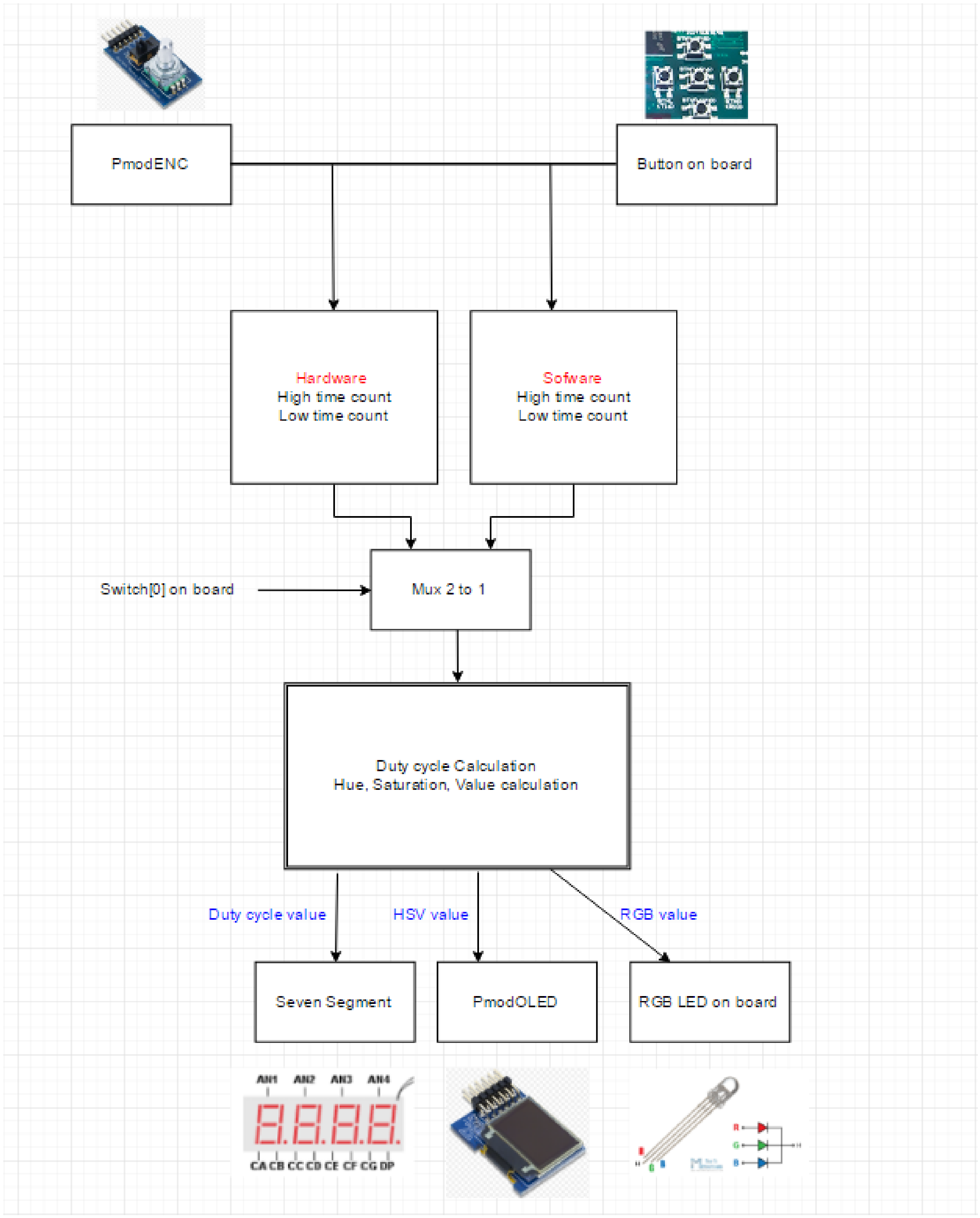
+ Enable Dual Channel

+ GPIO 2 width is 32 bits, all input is checked

+ Make the ports external and rename in terms of Red, Green, Blue (ex gpio_Red_H, gpio_Red_L).

These GPIO are inserted to send the hardware red, green, blue high time and low time count values to software.

3. Block Diagram



Functional specification:

+ Rotary Encoder (PmodENC): The rotary encoder knob on the PmodENC is used to alter the Hue value of the Colorwheel.

+ Pushbuttons:

1. System reset is done by pressing the CPU Reset button (the button is red) on the Nexys4 DDR. This button is mapped to the sysreset_n (asserted low) and sysreset (asserted high) signals in the top level
2. The btnR and btnL are used to vary the saturation value in the HSV scale
3. The btnU and btnD are used to vary the Value in the HSV scale

+ Slide Switch[0]: Slide switch on the NExys4 DDR selects between software and hardware pulse width detection. Hardware pulse width detection should be enabled when the switch is up and software pulse width detection should be enabled when the switch is down

+ The display of the PmodOLEDrgb:

1. Display of the current Hue, Saturation, and Value on the left side of the display
2. Display the rectangle showing the color for the corresponding value of the Hue, Saturation and Value in the HSV value

+ LEDs on the Nexys4 DDR: RGB1 and RGB2 the tri-color LEDs indicate the color selected from the colorwheel or the color displayed on the colorwheel. The signal driving the RGB display is taken for PWM detection.

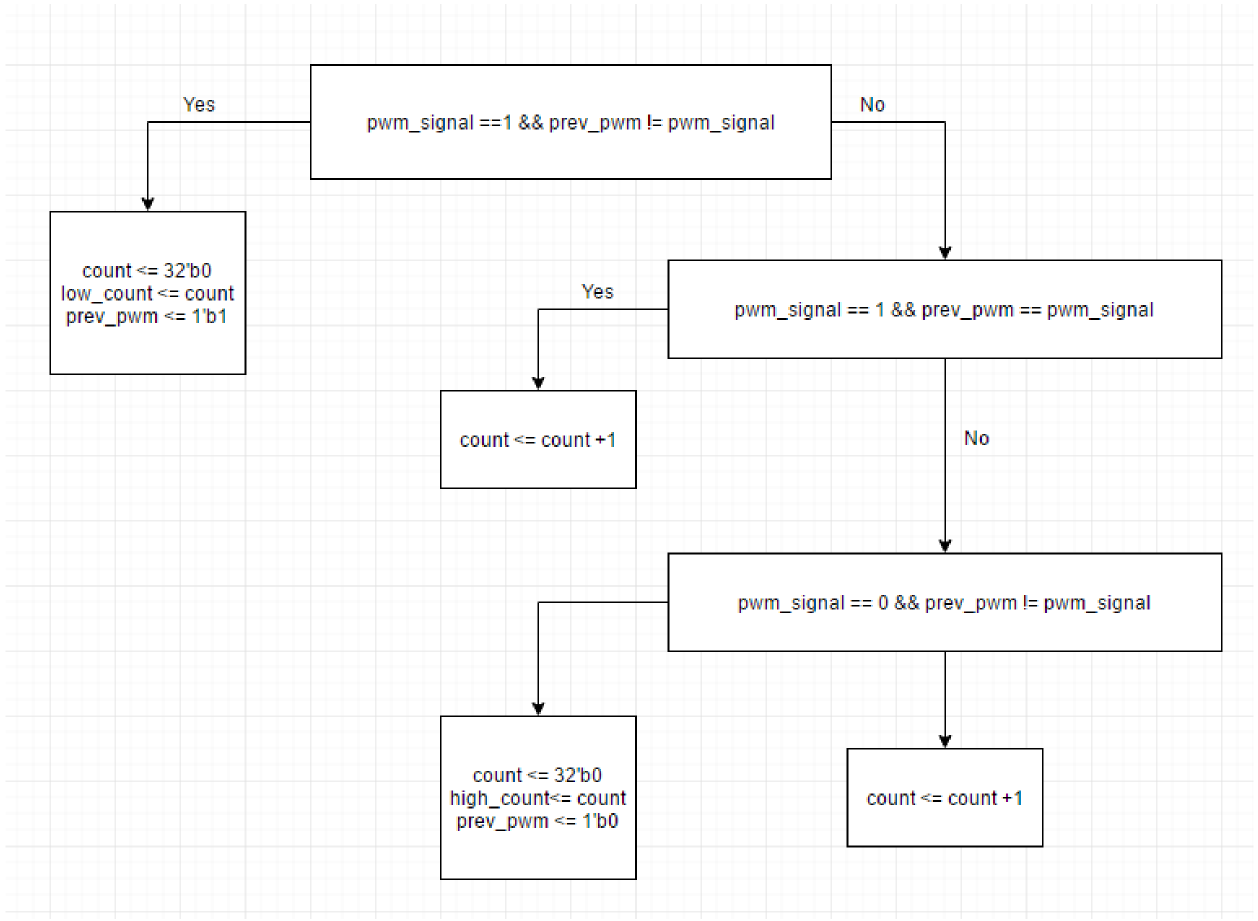
+ Seven Segment display on Nexys4 DDR:

1. Digit [7:6]: display the duty cycle detected for the RED RGB led
2. Digit [4:3]: display the duty cycle detected for the GREEN RGB led
3. Digit [1:0]: display the duty cycle detected for the BLUE RGB led.

4. Algorithm

a. Algorithm for counting high time values and low values in hardware:

The hardware module (high_low_count) provides a count of high and low interval time to the Microblaze. The inputs are AXI GPIO (PWM signal), clk, and reset signals. The outputs are 32-bit high_count and low_count. It implements a simple state machine to determine high-to-low and low-to-high transitions, and then store a counted value to one of two registers.



The counter for high and low time values depends on signal pwm_signal from AXI GPIO.

If pwm_signal == 1 and pwm_signal != prev_pwm (previous PWM): a transition from high → low. I reset value count, assign low count = count, and update previous PWM = 1. If no (pre_pwm == pwm_signal) I update count + 1

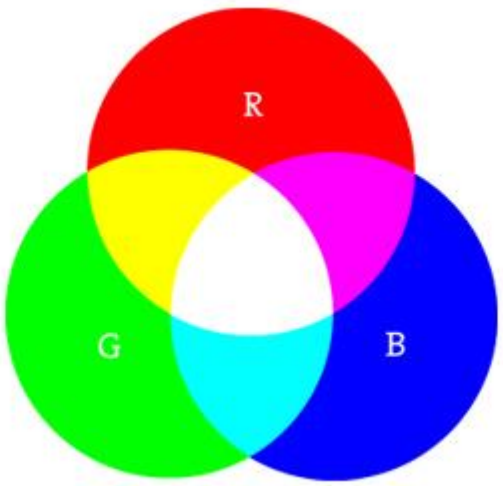
Similarly, if pwm == 1 and pwm_signal != prev_pwm: a transition from low → high. I reset value count, assign high count = count, and update previous PWM = 0. If no, I update count + 1.

b. Algorithm for counting high time values and low time values in software:

I use the same algorithm of counting high time values and low time values from hardware by checking transition low to high or high to low in order to determine the high and low value. These algorithm is done at function FIT_Handler (timer interrupt handler). However, a hardware pulse width detector can sense much higher frequencies than a pure software implementation. A software pulse width detector is simple to implement but cannot detect high frequency pulses. For example, the timer AXI timer has interrupt every 40KHz. If the frequency pulses is 50KHz, software pulse width detector cannot detect these signal. On the other hand, hardware pulse width detector can detect these signal.

5. Duty cycle calculation

I can control the brightness of an RGB LED by adjusting the duty cycle. I can control how much of each of the three colors I want in the mix of color by dimming them with various amounts.

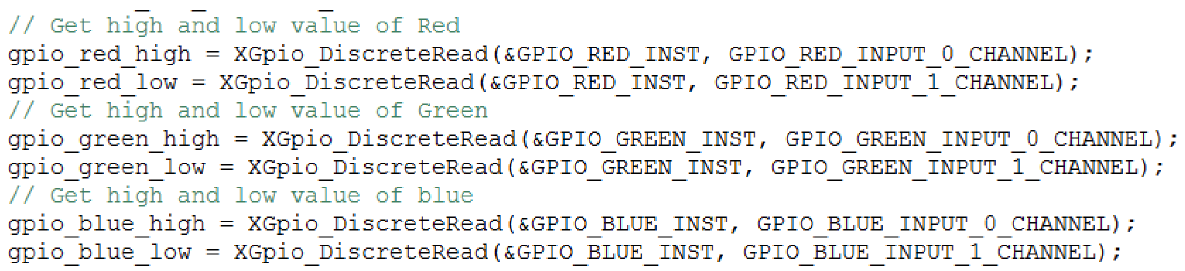


Basics of color mixing

(Reference: https://learn.sparkfun.com/tutorials/pulse-width-modulation)

If all three are on in equal amounts, the result will be white light of varying brightness. Blue equally mixed with green will get teal. As slightly more complex example, try turning red fully on, and green 50% duty cycle and blue fully off to get an orange color.

These duty cycle was calculated in software. The high count and low count in hardware were sent to software via additional axi_gpio:



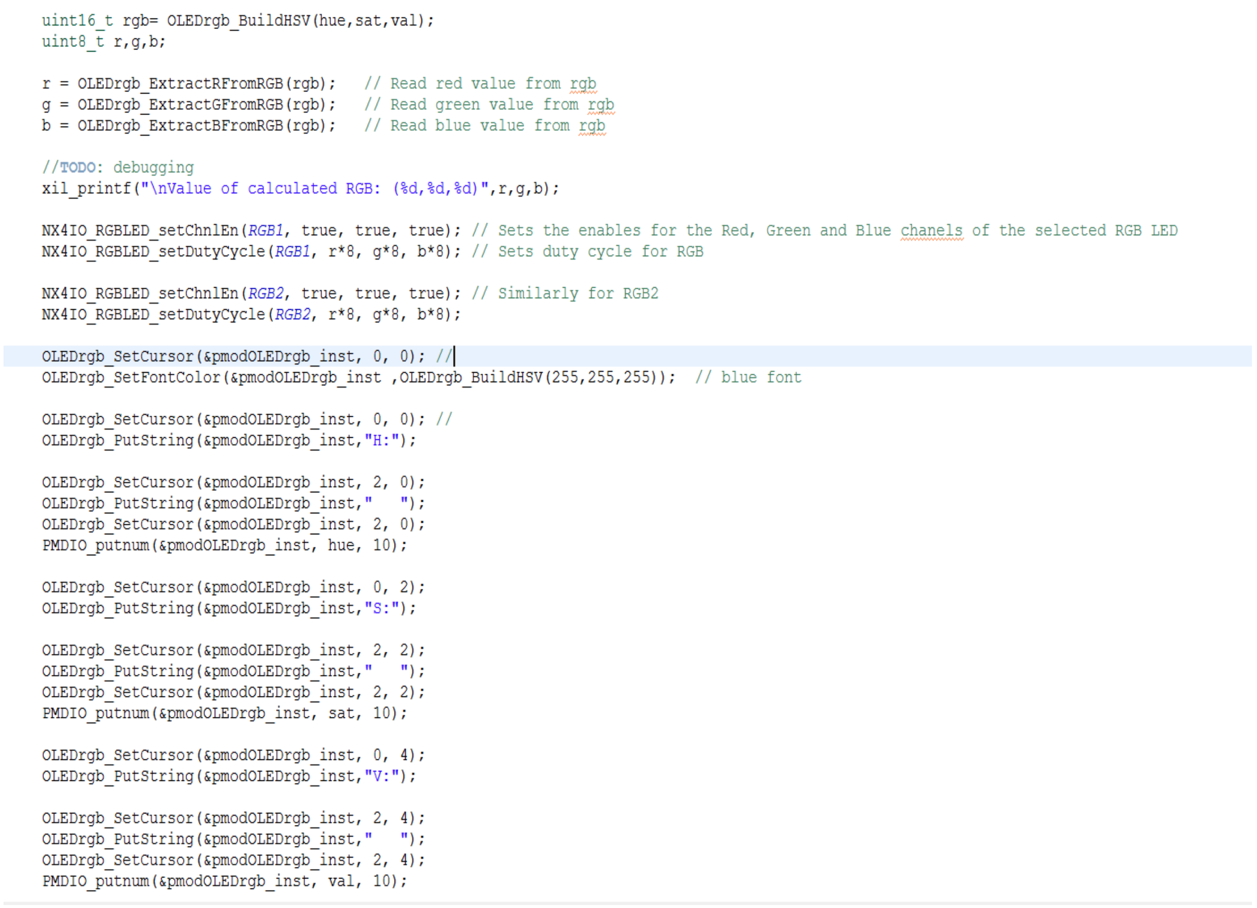
```
// Get high and low value of Red
gpio_red_high = XGpio_DiscreteRead(&GPIO_RED_INST, GPIO_RED_INPUT_0_CHANNEL);
gpio_red_low = XGpio_DiscreteRead(&GPIO_RED_INST, GPIO_RED_INPUT_1_CHANNEL);
// Get high and low value of Green
gpio_green_high = XGpio_DiscreteRead(&GPIO_GREEN_INST, GPIO_GREEN_INPUT_0_CHANNEL);
gpio_green_low = XGpio_DiscreteRead(&GPIO_GREEN_INST, GPIO_GREEN_INPUT_1_CHANNEL);
// Get high and low value of blue
gpio_blue_high = XGpio_DiscreteRead(&GPIO_BLUE_INST, GPIO_BLUE_INPUT_0_CHANNEL);
gpio_blue_low = XGpio_DiscreteRead(&GPIO_BLUE_INST, GPIO_BLUE_INPUT_1_CHANNEL);
```

The duty cycle was calculated as division of high time count and addition of high time count and low time count multiplied by 100.

Duty cycle = high time count *100/ (high time count + low time count). The duty cycle for hardware and software are maximum found to be 50%. These values were displayed on the seven segment display depending upon the value of switch[0] which if set to 1 displayed the hardware pwm values and vice versa.

6. Displaying the Hue, Saturation and Value on PmodOLED screen

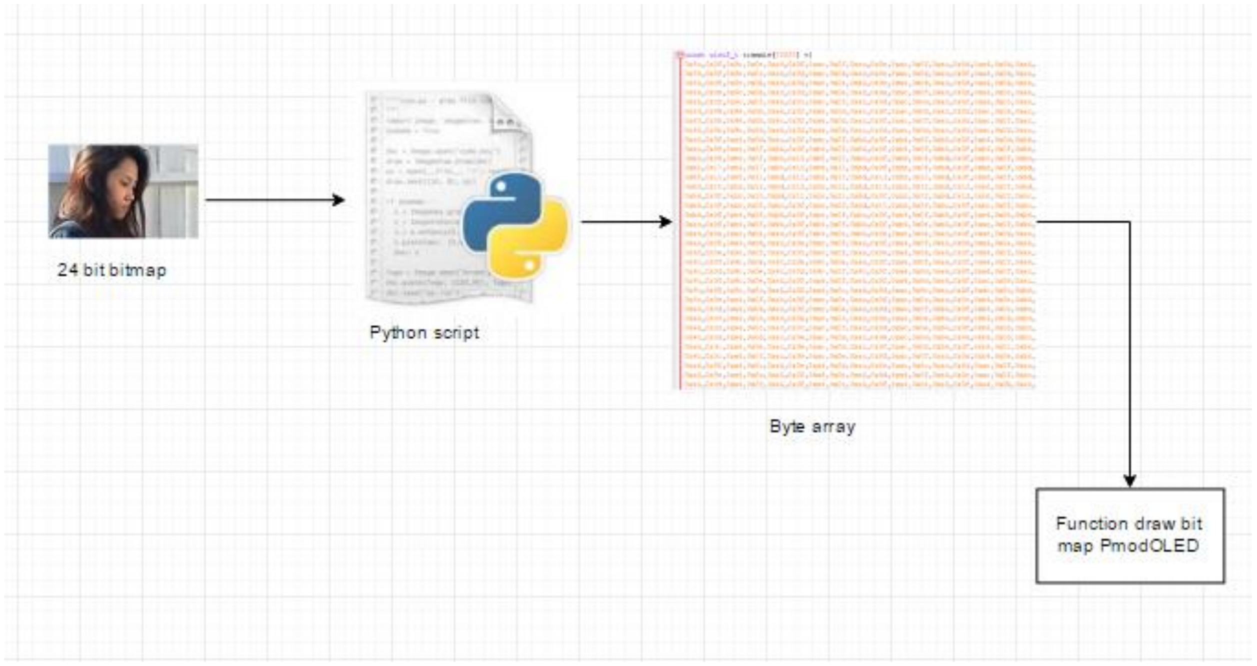
The PWM signal is sent to software via gpio pin which is 8 bit. We need to extract the red, green, blue colors signals from the PWM signal by shifting. The rgb signal is the combined value of all colors. The red, green and blue colors are extracted from the rgb signal and then according to that the duty cycle function is sent the individual red, green and blue colors signals to LED’s as well as the PmodOLED.



```
uint16_t rgb= OLEDrgb_BuildHSV(hue,sat,val);
uint8_t r,g,b;

r = OLEDrgb_ExtractRFromRGB(rgb);   // Read red value from rgb
g = OLEDrgb_ExtractGFromRGB(rgb);   // Read green value from rgb
b = OLEDrgb_ExtractBFromRGB(rgb);   // Read blue value from rgb

//TODO: debugging
xil_printf("\nValue of calculated RGB: (%d,%d,%d)",r,g,b);

NX4IO_RGBLED_setChnlEn(RGB1, true, true, true); // Sets the enables for the Red, Green and Blue chanels of the selected RGB LED
NX4IO_RGBLED_setDutyCycle(RGB1, r*8, g*8, b*8); // Sets duty cycle for RGB

NX4IO_RGBLED_setChnlEn(RGB2, true, true, true); // Similarly for RGB2
NX4IO_RGBLED_setDutyCycle(RGB2, r*8, g*8, b*8);

OLEDrgb_SetCursor(&pmodOLEDrgb_inst, 0, 0); //
OLEDrgb_SetFontColor(&pmodOLEDrgb_inst ,OLEDrgb_BuildHSV(255,255,255));  // blue font

OLEDrgb_SetCursor(&pmodOLEDrgb_inst, 0, 0); //
OLEDrgb_PutString(&pmodOLEDrgb_inst,"H:");

OLEDrgb_SetCursor(&pmodOLEDrgb_inst, 2, 0);
OLEDrgb_PutString(&pmodOLEDrgb_inst,"   ");
OLEDrgb_SetCursor(&pmodOLEDrgb_inst, 2, 0);
PMDIO_putnum(&pmodOLEDrgb_inst, hue, 10);

OLEDrgb_SetCursor(&pmodOLEDrgb_inst, 0, 2);
OLEDrgb_PutString(&pmodOLEDrgb_inst,"S:");

OLEDrgb_SetCursor(&pmodOLEDrgb_inst, 2, 2);
OLEDrgb_PutString(&pmodOLEDrgb_inst,"   ");
OLEDrgb_SetCursor(&pmodOLEDrgb_inst, 2, 2);
PMDIO_putnum(&pmodOLEDrgb_inst, sat, 10);

OLEDrgb_SetCursor(&pmodOLEDrgb_inst, 0, 4);
OLEDrgb_PutString(&pmodOLEDrgb_inst,"V:");

OLEDrgb_SetCursor(&pmodOLEDrgb_inst, 2, 4);
OLEDrgb_PutString(&pmodOLEDrgb_inst,"   ");
OLEDrgb_SetCursor(&pmodOLEDrgb_inst, 2, 4);
PMDIO_putnum(&pmodOLEDrgb_inst, val, 10);
```

7. Extra thing: draw picture (24-bit map) on PmodOLED screen

Diagram:



(Reference: https://reference.digilentinc.com/reference/pmod/pmodoledrgb/start#example_projects)

The diagram show step-by-step how I can draw picture 24 bit map on PmodOLED.

Because the size of PmodOLED screen only 96x64, I need to resize the picture (file bmp) to 96x64 to match the resolution.

Python script converts the 24-bit bitmap picture to byte array(These value represent pixel value on PmodOLED).

Using this function to draw the byte array on PmodOLED

```
OLEDrgb_DrawBitmap(&pmodOLEDrgb_inst,0,0,95,63,(uint8_t*)background);
```

This will draw my background array from (0,0) to (95,63).

Reference:

Digilent Nexys4 DDR Board Reference Manual. Copyright Digilent, Inc.

Digilent PmodCLP Parrallel LCD Reference Manual. Copyright Digilent, Inc.

Digilent PmodENC Reference Manual. Copyright Digilent, Inc.

Getting Started in ECE544 (Vivado/Nexys4) by Roy Kravitz https://en.wikipedia.org/wiki/HSL_and_HSV

https://learn.sparkfun.com/tutorials/pulse-width-modulation

https://reference.digilentinc.com/reference/pmod/pmodoledrgb/start#example_projects