ECE-544 PROJECT 1

Color Wheel Implementation using PWM Generation and Detection on XILINX NEXYS4 DDR.

JANUARY 28, 2018

AUTHOR: TEJAS CHAVAN

PORTLAND STATE UNIVERSITY

Email: techavan@pdx.edu

PSU ID: 975884622

A) ATTACHMENTS: The submission for the project includes:

1. Schematic of the Embedded system 'embsys.pdf'
2. Software Code: 'my_submit_code.c'
3. Hardware Code: n4fpga.v , pwm_hardware.v
4. Embsys schematic.
5. Readme file.

B) PROJECT DESCRIPTION: PWM Generation and Detection

This project was performed on the Digilent NEXYS 4 DDR board. The board is interfaced with the Pmod Encoder and the Pmod OLED display.

The FPGA board is programmed with Microblaze soft core which can be programmed using C. This project includes PWM Generation which is done by software (Programmed through Microblaze) and PWM Detection which is implemented in both hardware (Verilog Module) and software (programmed in Microblaze).

PWM Generation: The board takes inputs from the Rotary Switch of the Pmod Encoder (for Hue), from the Up-Down Switches (for Value) and the Right-Left Switches (for Saturation) and converts these values into RGB.

Functions used:

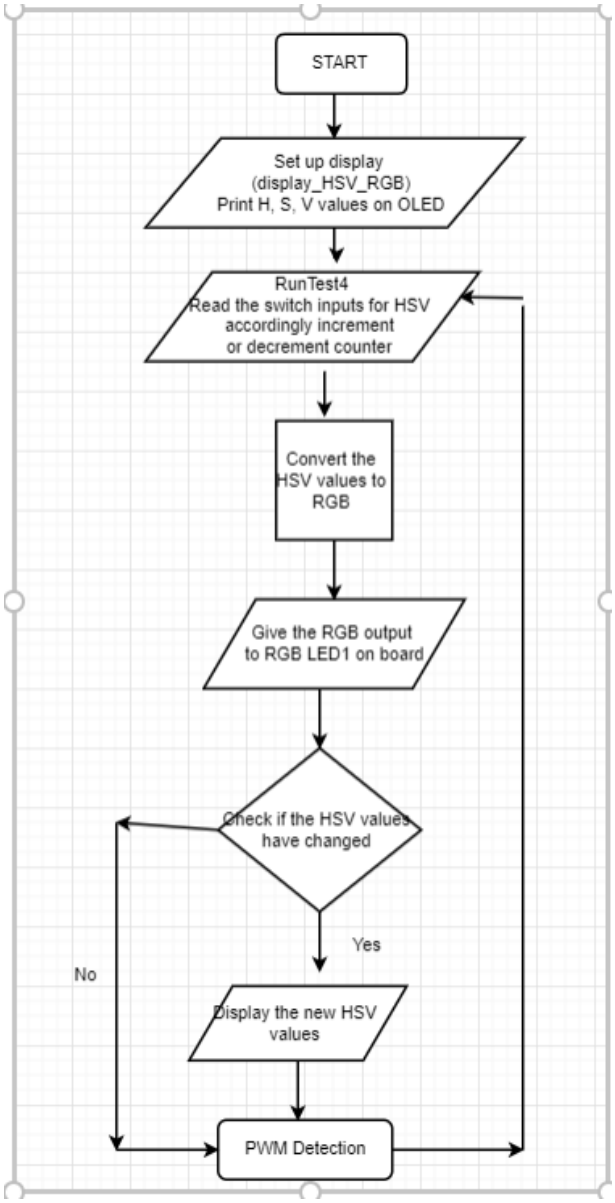
- **OLEDrgb_BuildHSV()** : The function returns a 16 bit unsigned integer which consists of 5 bits of Red, 6 bits of green and 5 bits of Blue.
- **OLEDrgb_BuildRGB():** Converts separate RGB values into a 565 RGB value used by the OLEDrgb. **(Note: While conversion, the function right shifts R by 3, G by 2 and B by 3. Due to this there is a loss of bits necessary to light the RGB leds. Thus we must left shift the return values in the next function to obtain the correct 8-bit R, G and B values)**
- **OLEDrgb_ExtractRFromRGB()** : Extracts the [Red,Green,Blue] value from the 565 color value.

The Hue, Saturation and Value are displayed on the OLED display along with the equivalent RGB values. Also, a Rectangle of the generated colour is drawn on OLED display. (Explained further below) .

PWM Detection: The generated RGB value is passed on to the RGB LED1 on the board. The PWM signals are also fed for PWM Detection (Hardware and Software). The hardware module (pwm_hardware) gets its PWM inputs from the signal coming to the RGB LED1. The software PWM Detection code gets its inputs from a GPIO Pin to the Microblaze which is fed again from RGB LED1.(Explained further below)

C) PWM GENERATION:

Flowchart:



- Three counters are initialised, RotaryCnt, Sat_Count and Val_Count. As the name suggests these counters are initialised to keep track of the HSV values.
- Every time the Pmod Enc is rotated, the value in the RotaryCnt is incremented. Similarly, the Sat_Count and Val_Count, are implemented to keep track of the button presses in Right/Left (increment/decrement) for Saturation and Up/Down (increment/ decrement) for Value. These counters are implemented. (using RunTest4).

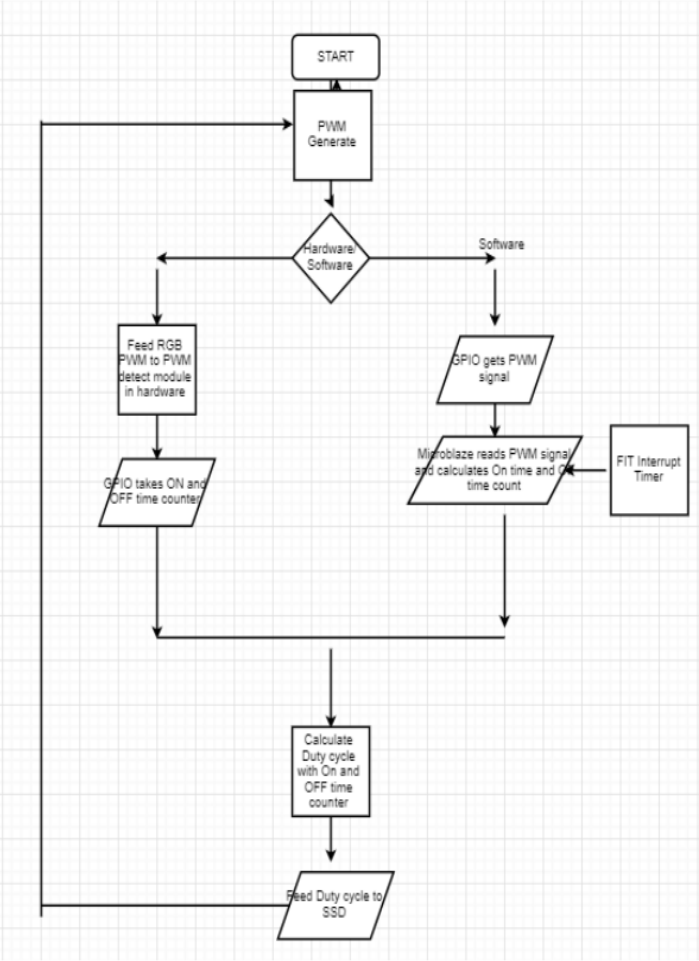
0-359 for RotaryCnt.
0-99 for Sat_Count and Val_Count Respectively.

- They overflow to 0 if incremented above top limit. These values are displayed on the OLED display. The HSV values are mapped down to 255 by multiplying each of these counters.
- For Hue, it is multiplied by (255/359) and for value and saturation, it is multiplied by (255/99).
- The scaling down is performed as the function to map HSV to RGB take 8-bit inputs of hue, saturation and value. These values are converted into RGB values of the format of 16 bits (5-Red, 6-Green, 5 –Blue using function OLEDrgb_BuildHSV()). This 16-bit value can be directly passed to the OLED display for displaying a rectangle along with the hue, saturation and value and red, green and blue values.
- These values must be separated and mapped back to a range of 0 to 255 to generate a PWM signal. (The function to set duty cycle of RGB LEDs takes input 8 bit for red, green and blue). This is fed to RGB
  LED1 on the board

NOTE: The values printing on the OLED will flicker in case they are displayed every cycle. Thus, they must be displayed only when it is different from the previous values. Also, when this is implemented with minimal delay, the SSD starts changing values too fast thus not giving a stable output. Thus a delay is needed in the entire loop to make the SSD values readable.

D) PWM DETECTION:

Flowchart:



1. HARDWARE PWDET:

- The Hardware Detection is performed using the verilog module 'pwm_hwd' and ‘pwm_hardware’. The 'pwm_hwd' module takes input as a pwm signal (either red, green or blue) and a 5MHz clock to sample the 4KHz PWM signal. The Code uses a simple logic of a counter which keeps on incrementing on posedge, negedge of pwm and on posedge of clock.
- The Hardware detection requires the addition of three more GPIOs each with dual channel configuration and 32 bits wide.
- The high_time and low_time calculated from the algorithm has to be instantiated to the new GPIO ports. Therefore 6 high and low counts => 6 GPIO ports.
- By applying edge detection algorithm the value of the count is assigned to high_count on every falling edge of the PWM signal. Similarly the count value is assigned to low_count on every rising edge of the PWM signal.
- This module is instantiated thrice, once each for Red, Green and Blue in ‘pwm_hardware’ .

```
always @(posedge clk or posedge color or negedge color)
    begin
        if (color==0 && old_pwm==1)                 //
            high_time=count;                        //
        else if (color==1 && old_pwm==0)
            low_time=count;                         //

        if (old_pwm!=color)
            count=15'd0;
                                                    //

        count=count+1;                              //

        old_pwm=color;                              //

    end
endmodule
```

- This is the basic logic that is replicated for RED, GREEN and BLUE.
- The calculated value of high_time and low_time needs to be passed to the microblaze for further calculation of duty cycle which is the same for software part as well.
- The high_time and low_time hold their value till new values are available.
- The high_time and low_time of all 3 PWM signals are passed to the software using 3 Dual channel GPIO ports each 32 bits wide for high and low time. The GPIOs are read using the function **XGpioDiscreteRead();**
- The PWM duty cycle is calculated by using the following formula:

Dutycycle = (High_time)*100/ (High_time + Low_time).

1. **SOFTWARE PWDET:**

- The Software PWM detection is performed inside the FIT handler module. The Fixed Interval Timer is basically an Interrupt mechanism that works on 40KHz frequency. Since the PWM is generated at 4KHz, the FIT handler provides interrupt 10 times in a one clock cycle of the 4KHZ PWM.
- The FIT module reads the GPIO input port by using the function **XGPIODiscreteRead()** and store the data in an 8 bit unsigned variable. In the n4fpga.v file we can see that, gpio_in is obtained from zero padding the 5 MSBs along with the one bit values for R, G B as LSB

gpio_in = {5'b00000, red, green, blue};

- Therefore we obtain the red value by performing "bitwise AND" on gpio_in with x0004 and right shifting by 2. Similar operations are performed on green and blue.
- After obtaining the signal, edge detection algorithm is performed on it. When the signal is changing from 0 to 1 the off-counter values are output to final off time and same is done for on time when signal changes from 1 to 0.

Note: When the interrupt is on at a very high rate, it gives better accuracy, but it causes the rest of the code to work slowly. Thus, the interrupts must be switched on only if in software mode and they must then be switched off after each time it samples.

The part ahead is common for hardware and software detection. The on times and off times are available. These are used to calculate the duty cycle of the PWM signal. This duty cycle is fed to RGB LED2 so that it can be compared with RGB LED1, for same values as the PWM generated.
The duty cycles are also displayed on the Seven Segment Displays (SSDs). These values can be compared with the RGB values on the OLED to compare the generated and detected values.

Functions used:

1. **XGpio_Initialize(XGpio * InstancePtr, u16 DeviceId):** Initialie the GPIO instance provided by the caller based on the givern DeviceID.
2. **XGpio_DiscreteRead(XGpio * InstancePtr, unsigned Channel):** Read states of the discretes for the specified GPIO.

Changes made in n4fpga.v and embsys:

- Added a PWM module(pwm_hardware) that performs Hardware Pulse Width Detetction.
- Created 3 new GPIOs (6 channels , 32 bit width each).
- Created a new clk_out 3 (5MHz) in clk wizard to use for to detect the pwm module.
- Connected the new RGB output from embsys to the pwm module.

MODULES USED IN BLOCK DIAGRAM :

1) Microblaze:- The MicroBlaze is a soft microprocessor core designed for Xilinx FPGAs from Xilinx. As a soft-core processor, MicroBlaze is implemented entirely in the general-purpose memory and logic fabric of Xilinx FPGAs.
2) Axi Timer:- The LogiCORE™ IP AXI Timer/Counter is a 32/ 64-bit timer module that interfaces to the AXI4-Lite interface.
3) Pmod ENC:- The Pmod ENC features a rotary shaft encoder with an integral push-button to provide multiple types of outputs such as a way for users to quickly switch between multiple choices shown on a screen or predefined motor speeds. The module also includes a sliding switch that is commonly used as an on/off output. An encoder is commonly used in freely rotating volume knobs to detect how many "clicks" a knob has been rotated.
4) PmodOLED :- The Pmod OLED is 128x32 pixel monochrome organic LED (OLED) panel powered by the Solomon Systech SSD1306. Users can display any sort of graphical design by programming the device through SPI as well as sending bitmap images.
5) AXI GPIO :- The AXI GPIO provides a general purpose input/output interface to the AXI (Advanced eXtensible Interface) interface. This 32-bit soft IP core is designed to interface with the AXI4-Lite interface..

BASIC FLOW FOR THE PROGRAM BASED ON FUNCTIONS:

1. **Main():** The main function where the program begin contains function calls are according to proper flow
2. **Init_platform():** Initializes all the APIs used in the program.
3. **FIT_Handler():** This function basically acts as an interrupt and reads the gpio for software detection of PWM duty cycle. This is were the software detection is implemented.
4. **Microblaze_enable_interrupts():** Starts the timer and kick starts the processing by enabling the microblaze interrupts.
5. **Display_HSV_RGB():** Displays the initial text and initialized HSV values on the PMODOLED.
6. **HSV_RGB():** This is the major function that performs most of the operations including HSV to RGB conversion, PWM Generation , software and hardware PWM detection.
7. **pwm_hw():** Calculates the duty cycle of the PWM signal using HW detection only if the new value is different than the old value.
8. **pwm_sw() :** Calculates the duty cycle of the PWM signal using SW detection only if the new value is different than the old value.
9. **Display_OLED():** Displays the new HSV values on the OLED.

CHALLENGES FACED AND SCOPE OF IMPROVEMENT:

Understanding the drivers: Though the functions were well documented, understanding their operation and flow took a lot of time.
I could have planned the project in a much better way. Though I went on building it incrementally, I faced a few issues in understanding a few major functions.

Overall It was a very good experience working on system level and I hope to put better efforts in the upcoming projects.

REFERENCES:

1) Getting Started Manual.
2) Nexys4 board Xilinx manual
3) Xilinx forums and Stackoverflow
4) https://www.xilinx.com/products/intellectual-property/axi_gpio.html
5) https://www.xilinx.com/support/documentation/ip_documentation/axi_timer/v2_0/pg079-axi-timer.pdf