ECE 544 EMBEDDED SYSTEM DESIGN

Project 1 Report

Surya Ravikumar

19th April 2019

Project Summary

The purpose of this project was to gain experience generating an embedded system application using the Xilinx Vivado and SDK embedded system tools. The main requirements to be fulfilled were to

1. Add functions to generate inputs for the RGB LEDs on Nexys A7 using Pulse Width Modulation (PWM). The duty cycle of the signal is determined by the user input (using a rotary encoder and pushbuttons on the board). PWM of LEDs is then fed back to the embedded system via a GPIO input port for hardware and software pulse width detection.
2. Detect pulse width of the three signals in software by reading 'the' GPIO port and calculate the duty cycle in software
3. Detect pulse width of the three signals in hardware by using the signals fed back and calculate duty cycle in hardware.

Functional Specification

Firstly, a variable duty cycle PWM output signal for the on board RGB LEDs had to be generated. As each color in the color spectrum/circle had a respective RGB value, the RGB value would decide the duty cycle to be written. Further, the user would decide what color to output using the PmodENC and pushbuttons. However, the user input would be in the HSV format (a color wheel) which had to be converted RGB before generating these signals. Turning the encoder right had to increase 'hue', and turning it left decreased 'hue'. The left and right buttons decreased and increased 'saturation' respectively. Up and down buttons increased and decreased the 'value' parameter respectively. The HSV values, and the respective color should also be displayed in the PmodOledRGB. Left side of the display displayed the current hue in degrees, saturation & value from 0 to 100, and a rectangle box on the right displayed the respective color.

Secondly, the generated PWM signal that is written to the LEDs if fed back to the system as inputs via a GPIO port to detect the duty cycle. Software and Hardware algorithms were implemented to detect the duty cycle of the 3 signals. Software detection uses the GPIO port to read these signals while the hardware module uses the signals written to the LEDs directly. The duty cycles of the signals were to be displayed on the seven segment displays on board - red's duty cycle in digits 7 & 6, green's duty cycle in digits 4 & 3, and blue's duty cycle in digits 1 & 0. Also, slide switch 0 selected between hardware and software detection – on/1 for hardware detection and off/0 for software detection. LED 0 was also lit of hardware detection was selected and unlit for software detection. The program had to seamlessly switch between hardware and software detection while running.

Finally, the center button and pushbutton on the encoder had to terminate the program, and the reset button had to reset the board.

Design

User input

When the program starts executing, or is reset, the hue, saturation and value parameters are set to 0 degrees, 100, 100 respectively. Hue is controlled using the rotary encoder – right rotation increases hue and left rotation decreases it. Hue value can go from 0 to 359. Once it reaches 359, it wraps back to 0, and once it reaches 0 when decreasing, it wraps back to 359 instead of going to negative 1. Saturation is controlled using left and right pushbuttons – left decreases it and right increases. Once saturation reaches hundred, it stays at 100 even if the right pushbutton is presses, and once it reaches zero, it stays at 0 even if the pushbutton is pressed. Value is controlled using up and down pushbuttons – up increases it and down decreases it. Like saturation, value is set to not go beyond 0 and 100. These values are displayed to PmodOledRGB using given functions to set cursor, print numbers and text.

HSV to RGB

HSV values are converted to RGB values from a 0 to 255 scale. To do this, a new function was created which took in H, S, V parameters and returned R, G, B parameters. Saturation and Value were scaled down such that it was a fraction in the range [0, 1]. The R, G, B values, in [0, 1] scale, were calculated the following way:

$$C = V \times S_{HSV}$$

$$H' = \frac{H}{60^\circ}$$

$$X = C \times (1 - |H' \mod 2 - 1|)$$

$$(R_1, G_1, B_1) = \begin{cases} (0, 0, 0) & \text{if } H \text{ is undefined} \\ (C, X, 0) & \text{if } 0 \leq H' \leq 1 \\ (X, C, 0) & \text{if } 1 < H' \leq 2 \\ (0, C, X) & \text{if } 2 < H' \leq 3 \\ (0, X, C) & \text{if } 3 < H' \leq 4 \\ (X, 0, C) & \text{if } 4 < H' \leq 5 \\ (C, 0, X) & \text{if } 5 < H' \leq 6 \end{cases}$$

$$m = V - C$$

$$(R, G, B) = (R_1 + m, G_1 + m, B_1 + m)$$

Figure 1 HSV to RGB Source: Wikipedia

R, G, B values are then scaled to [0, 255] by multiplying them by 255. These RGB values were then used in displaying a color box to PmodOledRGB with the help of a given function to build a rectangle. These RGB values were also written to the onboard RGB LEDs using given functions.

Software Pulse Width Detection

RGB values written to the LEDs were fed back to the system via a GPIO port. This port was read in the software code during every iteration, if software pulse width detection was selected. The bottom 3 bits of the register value read were masked out to get R, G, B signals and specific R, G, B counters were incremented if the signals were a logic high (1). A counter variable (to count period) was also incremented every iteration. Once the counter variable reached the preset period, duty cycle of all three signals were calculated by just dividing the RGB counters with the total count – high counts/total counts in essence. However, the nexy4io module limits the PWM values to max out at 50%. So, the high count had to be multiplied by 2 in order to our duty cycle result to be in the range 0 – 99. After the duty cycle was determined, they were displayed on the seven-segment display. The program then continued by resetting the counters (start of new period) and accumulating counts from next iteration.

Hardware Pulse Width Detection

A new module was created in hardware with the following inputs: a new 5 MHz clock, global reset, red signal, green signal, blue signal (signals fed back from RGB LEDs); and the following outputs: red LED high counts, green LED high counts, blue LED high counts, total counts (period). This module was instantiated in the top module. On every posedge of clock, the total count would be incremented, and if the red, green or blue signals were sampled high, their respective counters would be incremented. Once the total count counter reached a predetermined value, the RGB counters would be assigned to output ports. However, due to unexplained errors where the output ports were not assigned properly, this was changed such that the RGB counters were directly assigned to the output ports instead of waiting till the count reached the predetermined value. Once count reached the predetermined value, all the counters were reset.

To pass on these values to the software module, two new GPIO ports were added to the embedded system and these values were passed on as inputs to these GPIO ports. The software, if hardware pulse width detection was selected, would then read these ports. The duty cycle was calculated ONLY IF the total count value passed in was close to the predetermined count to overcome the unexplained issue stated. Before the duty cycle was displayed, the high counts had to be multiplied by 2 again to scale the result from 0 – 99.

Observations

While implementing software pulse width detection, the final duty cycle was also half of what was the right value. After consulting Digilent website and documents, I realized I had to multiply the value by 2 since duty cycle of PWM signal written to the LEDs were capped off at 50%.

Initially, I had tied up my hardware PWDET module to the embedded system. Again, due to an unexplained issue, the total counter did not increment even though the RGB counters incremented. Therefore, I had to remove the module from embedded system and instantiated it in the top module. After instantiating in the top module, the RGB counter values were not assigned to the output ports when the total counter reached the predetermined value. Therefore, I assigned the counter directly to the output port to make it work (and calculated the duty cycle in software only if the total counter was within some region of the predetermined value).

From my experience, the software detection was more stable, but the hardware detection was more accurate. However, both methods always got the correct duty cycle within ±3 range.