Andrew Mikhail

CMPE 121L: Microprocessor System Design Lab

Lab Exercise 5

13 November 2016

Section: Tuesday 10-12pm

Table of Contents:

Introduction:

The purpose of this lab is to familiarize students with using the 16x32 RGB LED display. More specifically, we designed a controller for the LED display to toggle the 1536 LEDs available for us to use. We implemented the LED controller using software interrupt driven refresh as well as a designed hardware controller. Both methods were employed with row-column multiplexing in mind. Though, we only used a 4x4 grid of LEDs in this lab.

Prior to starting the lab we first had to understand how to connect the LED display to our PSoC 5 board. The LED panel required 5 Volt operation and a precise wiring connection with the PSoC 5 board. We also had to become familiar with how the LEDs are organized and how to display a pattern. To do so, we had to learn the flow of bits from PSOC to the LED matrix. Although each step was highly technical, we performed a number of basic tests on the LED panel to become comfortable with interfacing the hardware.

<u>Part 1: interrupt-Driven Refresh</u>

In this part of the lab we toggled the LEDs using a scan within an interrupt. Throughout this lab multiple LEDs always appear to be on at the same time. However, the rapid scan and refresh of each row of the LEDs made it seem that multiple rows are concurrently on. I implemented the refresh by creating an interrupt from a timer every 1ms. I chose a 1ms interrupt because that was an adequate speed for the human eye to not notice the refresh.

The colors of the LEDs were set in 3 two-dimensional arrays. For example if the index of red[i][j] is set then the red LED at row i column j would be on. Likewise, the two other LED arrays, green[i][j] and blue[i][j], controlled the green and blue LEDs of the panel. All three color arrays, red/green/blue, were initialized to zero and then manually overwritten to a specific color pattern in the main function. Moreover, I wrote the colors selected during the refresh to an RGB control register that set the GPIO pins for color. Likewise the clock, latch, and output enable pins all had their individual control registers that I wrote to in order to set the GPIO pins.

Within the interrupt I first selected the row I would be toggling. I then determined the color by checking the set bits of the red, green, and blue arrays. I then pulsed the clock pin from high to low, which set that bit for the selected column. I also pulsed the latch pin to shift the data into the shift register. I finally set the output enable pin to low, which enabled the display. I then incremented the row and looped again until all 4 rows and 32 columns had been scanned.

Part 2: Hardware Controller

In this part of the lab, we had to implement an LED hardware controller that would perform the same functions as Part 1. Unlike Part 1, the state of the LEDs was stored in control registers rather than memory arrays. More specifically, each control register had three outputs corresponding to the RGB color choice of that single LED. Thus, I had 16 control registers for each LED for the 4x4 grid that I would be using. The outputs of the control register were then first multiplexed by column then by row to determine which LED is selected. The output is then written to the RGB GPIO pins.

In order to select by row/column, I also had row and column counters to cycle through the positions of the LED matrix. I then pulse the latch, and output enable pins to authorize the display on the panel once I had cycled through all the columns. I also connected my clock pin to a 10kHz system to ensure the LED refresh would not be visible.

Part 3: PWM Control of LED Brightness

In this section of the lab, we had to improve on the design of Part 2 to allow 256 levels of LED brightness. To do so, I added an analog-to-digital converter (ADC) to receive analog readings from the onboard potentiometer. I also added a PWM whose output would be used a select line for a multiplexer to determine the level of brightness of the colors. Thus, if the potentiometer was turned to its rightmost position, then the LEDs would be at their brightest. Likewise, if the potentiometer were turned to its leftmost position, then the LEDs would be turned off. I also increased the clock frequency in this section to 256kHz to avoid light flicker on the LEDs at low-level brightness.

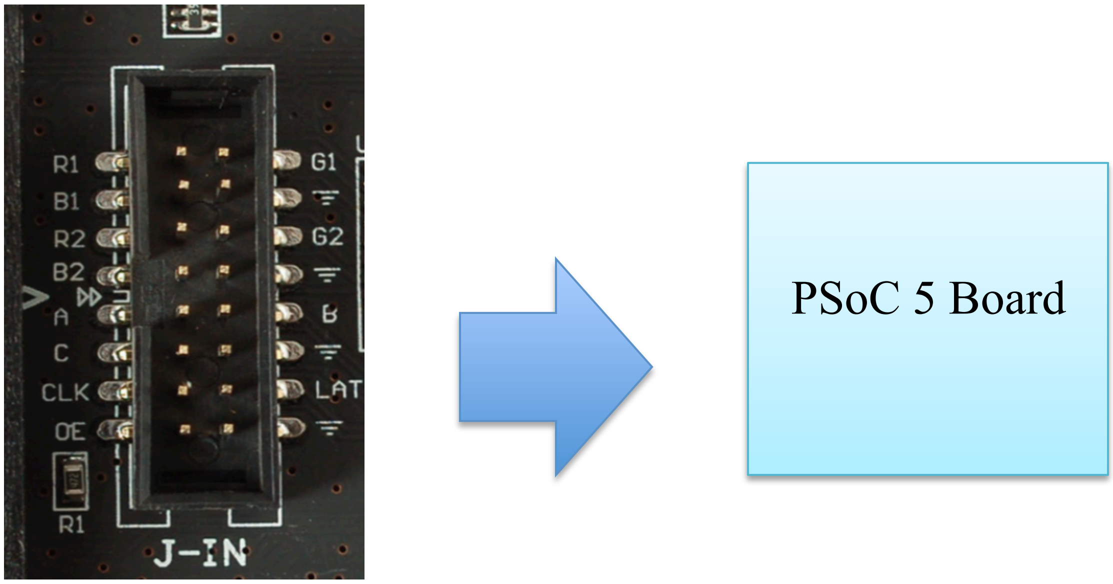
Conclusion:

This lab was implemented using the software interrupt-driven refresh method and the hardware controller method. The hardware method is significantly more tedious to setup than the software refresh. However, after the initial design, it is more efficient and predictable than the software interrupt method. Another disadvantage of the hardware method is that it is more challenging to create a variety of patterns. On the other hand, the software method is marginally easier to implement. Moreover, the software interrupt-driven refresh allows for simple implementation of more complex patterns on a larger scale. However, I would imagine that this method is less efficient than the hardware method since it utilizes software on top of the internal hardware. Nonetheless, I will most likely use the software refresh method in my final project because of its simplicity to work with.

One of the limitations of these two methods is that all the LEDs have the same duty cycle (1/8). This limits the number of color shades available to 8. Though, adding duty cycle adjustability should allow me access to the 4096 different colors. I could enhance Part 2 by adding a PWM for each color to adjust the duty cycle of each LED.

I thought this lab was effective in showing us how to interface with the LED panel. This lab provided a complete introduction to the hardware of the LED matrix as well as toggling different LEDs on the panel. I also became more aware of power restrictions associated with connecting devices to our PSoC 5 board. Moreover, this lab provided us an opportunity to build a variable hardware controller for the LED matrix out of logic primitives. Overall, I thought this lab was helpful in establishing a concrete understanding of controlling the LED panel. However, I think that this lab can be improved by giving students a more detailed explanation of the wiring of the LED panel. I had a great deal of trouble setting up the hardware and I think a better explanation of the explicit wiring connection would benefit future students.

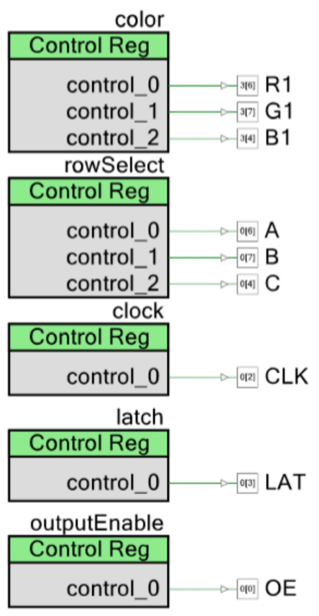
Appendix A: External Schematics

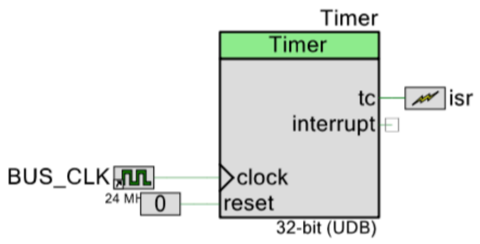


LED Panel Pin	PSoC 5 GPIO Pin
R1	P3[6]
B1	P3[4]
A	P0[6]
C	P0[4]
CLK	P0[2]
OE	P0[0]
G1	P3[7]
B	P0[7]
LAT	P0[3]

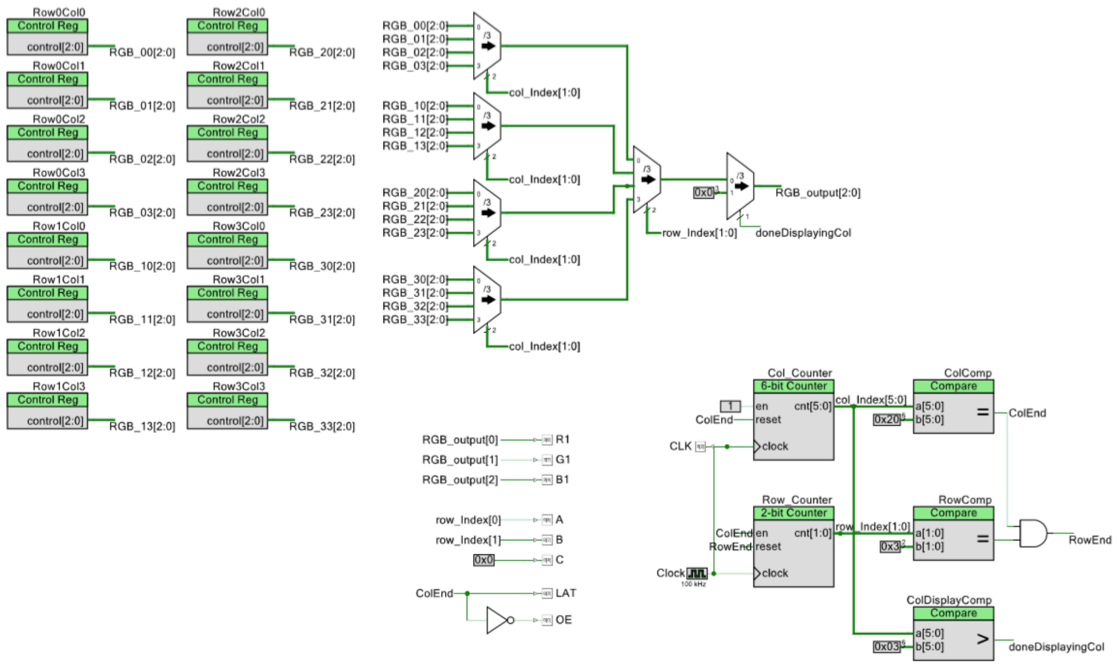
Appendix B: Internal Schematics

Part 1:

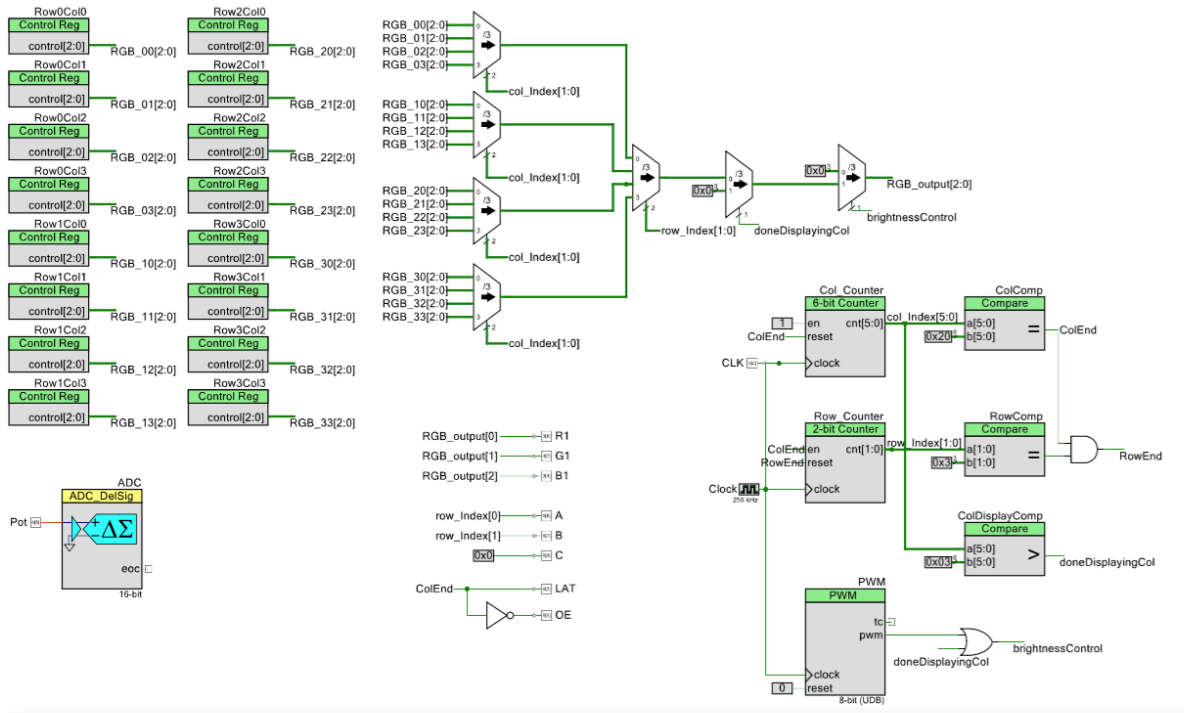




Part 2:



Part 3:



Row0Col0
Control Reg
control[2:0]
RGB_00[2:0]
Row0Col1
RGB_01[2:0]
Row0Col2
RGB_02[2:0]
Row0Col3
RGB_03[2:0]
Row1Col0
RGB_10[2:0]
Row1Col1
RGB_11[2:0]
Row1Col2
RGB_12[2:0]
Row1Col3
RGB_13[2:0]
Row2Col0
RGB_20[2:0]
Row2Col1
RGB_21[2:0]
Row2Col2
RGB_22[2:0]
Row2Col3
RGB_23[2:0]
Row3Col0
RGB_30[2:0]
Row3Col1
RGB_31[2:0]
Row3Col2
RGB_32[2:0]
Row3Col3
RGB_33[2:0]
col_Index[1:0]
row_Index[1:0]
doneDisplayingCol
0x0
RGB_output[2:0]
brightnessControl
Col_Counter
6-bit Counter
en
reset
clock
cnt[5:0]
col_Index[5:0]
ColEnd
CLK
ColComp
Compare
a[5:0]
b[5:0]
0x20
Row_Counter
2-bit Counter
cnt[1:0]
row_Index[1:0]
RowEnd
Clock
256 kHz
RowComp
a[1:0]
b[1:0]
0x3
ColDisplayComp
0x03
PWM
tc
pwm
8-bit (UDB)
ADC
ADC_DelSig
Pot
eoc
16-bit
RGB_output[0]
R1
RGB_output[1]
G1
RGB_output[2]
B1
row_Index[0]
A
row_Index[1]
B
C
LAT
OE