Driving an RGB LED Package with Variable Duty Cycle to Produce a Large Range of Colors as Perceived by an Observer

Ashley Turner, Kurtis Craig, Malcolm Whitehouse
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: klcraig@oakland.edu, anturner@oakland.edu, mewhiteh@oakland.edu

***Abstract*—RGB LEDs are an up-and-coming technology that can be used to create, in theory, any color in the spectrum of visible light. In this project, we took the opportunity to familiarize ourselves with this technology and learn about the implementation and application of RGB LEDs, as well as other related topics. In addition, we learned about how color is measured and how to work with RGB LEDs effectively.In addition, we learned from our mistakes in implementation that time-based functions take great care to implement well, and that it is important to know the limitations of the technology one is working with.**

I. INTRODUCTION

This project was chosen as an opportunity for us to learn about the implementation and applications of RGB LEDs. LEDs in general, as energy-efficient as they are, are taking over in a big way. With such a large number of lighting applications from street lights to house-hold light bulbs being replaced with LEDs, it has become clear to us that, as electrical engineers, any knowledge of and experience with LEDs that we can accomplish will go a long way in our careers and in our daily lives.

This project in particular covers some fundamental points on LED control. In the process of implementing Mode 1, where the color displayed by the RGB LED is controlled by the values on the DIP switches, we learned that several timer-based functions in conjunction can be tedious to implement. We learned improved methods for the use of timers and the dangers of irresponsible programming. With respect to the function of this mode on its own, the possible applications are many. Some examples of the application of user-controlled RGB displays include, for example, solar-powered lawn lamps of which the user may change the color for holidays or however they please.

In Mode 2, we made use of our knowledge of the PWM channels and the ADC functions to control the color of an LED with input from a sensor. The possibilities are virtually endless when it comes to sensor-controlled RGB LED applications. One possible example, as mentioned in the presentation, would be to mount RGB LEDs in parking lot lights, indicating by their color whether the parking spot is taken or available.

In Mode 3, the color displayed on the RGB LED was controlled by a joystick (Originally an accelerometer. We will cover this in the results section.). The user is asked to imagine that they are controlling a curse on the CIE 1931 xyz Chromaticity Diagram, and as he/she changes the position of the joystick, the color displayed by the RGB LED corresponds to the color coordinates of the cursor's position.

For us at this time, implementing this mode was a great exercise in learning, intuitively, the interactions between colors and gaining an understanding of this widely-accepted measurement standard. In the real world, an optimized version of this device could be used as a validation tool in the process of engineering parts that contain LEDs. If such a device could be programmed by a user to display any of the possible color coordinates, then a developer or customer could use this device to validate products against their color requirements.

II. METHODOLOGY

A. Hex Keypad

The hex key pad code is the main driver for the different modes, using this code we would be able to switch between one of 3 modes by simply pressing the assigned button for that mode. The hex key pad code has several things going on, we will not talk about his due to report length limits. When integrating the code for the other modes (starting with mode 1) into the code for the hex key pad we encountered a major problem. We suspect that the delay function used to operate the LCD display is also used to run some of the timer PWM signals for the RGB LEDs. This issue created problems with controlling the RGB and LCD so the program would get stuck. We attempted several solutions to try to fix the issue but none worked, due to time constraints were not able to include this in the final presentation.

B. RGB LED Strip

The RGB LED strip was an idea to get more noticeable color difference due to the higher output LEDs and number

of LEDs used. While you can buy RGB LEDs in many packages the LED strips are commonly available. An issue with this setup is that these strips are designed to run at 12v and the HCS12 only puts out a logic voltage level of 5v. Another issue was even though we had more RGB LEDs and the light output was higher, how would we best "display" theses colors. This is when we decided we needed a diffuser to better mix the colors.

C. Mode 1 –Controlling the RGB with DIP Switches

In Mode 1, the eight DIP switches control the color displayed on the RGB LED. The logic used to display the colors is modeled after the way VGA calls out colors using eight bits. DIP switches 1-3 control red, switches 4-6 control green, and 7-8 control blue. We used case statements to change the value of "HCYCLES" and "LCYCLES" for each LED, where the duty cycle is proportional to the binary value of the two or three DIP switches controlling the color. This means that there are eight different duty cycle values for red and green, and only four different duty cycles for blue.

In this mode, the LEDs are driven using timer interrupts. This code was modeled after the example code "unit9c.c" uploaded on Moodle and used functions from the "timers.h" file provided to the class by Lincoln Lorenz. Timer channels 1-3 were used in this case to call interrupts that toggled their corresponding bit on Port P, which in turn toggled the signal on each color of the RGB LED. The "main" portion of the code in Mode 1 constantly checks the DIP switches, changing the number of clock cycles between the toggles of each bit accordingly, as previously states.

D. Mode 2 – Temperature-Controlled Colors

In Mode 2, the color displayed by the RGB LEDs was controlled by the temperature reading from the on-board temperature sensor. The goal, originally, was to create a smooth fade between blue at cold temperatures, white at room temperature, and red at warm temperatures. Unfortunately, this smooth transition proved impossible. As it turned out, the temperature sensor on the development board had a resolution of only 1°C. In order to create the smoothest transition, then, we found the largest range of temperature that we could reach during a demonstration in class, and based the calculations of the duty cycle of each color on this range of temperatures.

We determined the definition of "cold" and "warm" based on the temperatures we could reasonably reach in demonstration. By holding a frozen lunch box cooler to the board, we found we could reach 16°C, and we could reach 43°C by applying heat from a dryer to the temperature sensor.

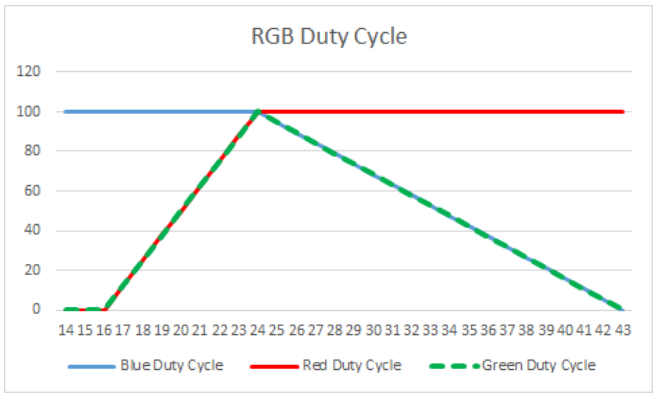

Figure 1- Duty Cycle with Respect to Temperature

The interesting problem in this mode was determining how to fade from blue to white, and from white to red. As it turns out, in additive color mixing, light blue can be created by driving red and green at the same duty cycle as one another, but with a lower duty cycle than blue. The greater the difference between the duty cycle of blue and the duty cycle of red and green, the closer the displayed color will be to blue. As the duty cycle of red and green increases towards the duty cycle of blue, the color shifts more towards white. The same concept applies to fading between white and red by holding red at 100% duty cycle, and decreasing the duty cycle of blue and green together as the temperature increases. We discovered the properties of additive color mixing by experiment, changing the duty cycles of the LEDs until we gained an intuitive understand of the result of mixing different colors of light together.

In Mode 2 and Mode 3, we decided to drive the LEDs using the PWM channels for simplicity in programming. The duty cycle, calculated using the algorithm described above, would be set continuously in an infinite loop, using a ratio of the number of clock cycles that make up the entire period. These calculations were carried out in a function that accepted two arguments: Color and duty cycle percentage. "Color" was simply an integer, 0 -2. We declared three integers that related "RED," "BLUE," and "GREEN," to these integers, so an example of such a function call would look like "set_PWM(RED, 50)."

$$PWMDTYn = \left(\frac{Cycles\ in\ Period}{100}\right) \times Duty\ Cycle\ \%$$

E. Mode 3 – Navigating the CIE Chromaticity Diagram

The reference for the colors to be created is the CIE Chromaticity standard graph. CIE is the International Commission on Illumination and the body responsible for lighting and color standards around the world. The graph in figure 2 shows the range of visible colors as can be seen by the human eye. The Y-axis is the hue and the X-axis is the measure of color saturation. The triangle portion of the

graph is called the gamut and represents the colors that can be reproduced correctly with RGB LED's. This also explains the color variation in the accompanying video of the project as the graph is referenced outside this area.

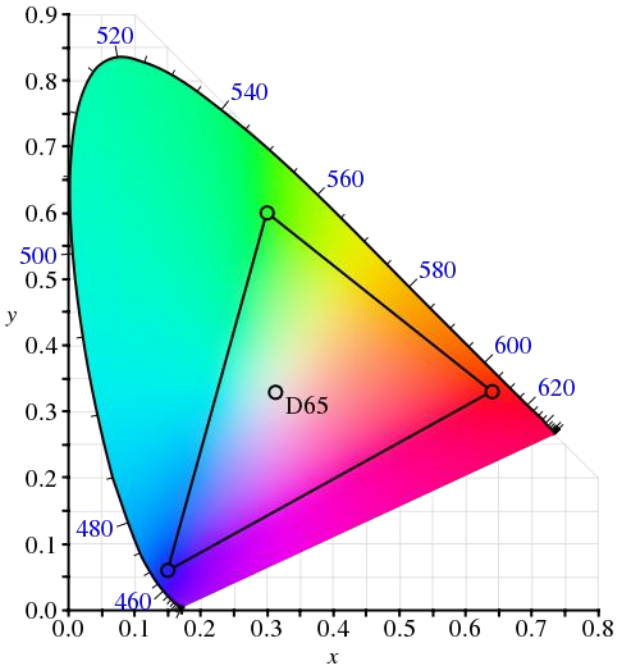

Figure 2 - 1931 Chromaticity Diagram from Reference [5]

Originally, we had planned on using an accelerometer for this mode, but due to the noise of the unit (failure of the accelerometer) we chose to try some different hardware as a last-ditch effort to test whether this noisy information was the fault of our design, or an unavoidable caveat of the accelerometer. This new hardware, an analog, dual-axis joystick, produced an extremely clean and reliable signal. This resolved our problem with this mode, and since then we have had no problems with this code. Naturally, the formulas calculating the duty cycle of each color at all points in the color space could use plenty more optimization, but the code does work for our purposes.

In addition, the color coordinate of the color currently displayed on the RGB LEDs were also displayed on the LCD. We found that the color displayed on the RGB LEDs was very close to the corresponding color coordinates on the actual CIE 1913 Chromaticity Diagram, meaning that we had reached our objective for this mode.

The first thing we needed to do was translate the position of the joystick to a position on the color space. Using the debugger feature of the CodeWarrior IDE, we found the integer values of the up and down position and the left-right position at each extreme in the x and y directions of the joystick. Then, we simply used a ratio to convert these values into color space values. For example, in the x direction, we found that we got 0 when the joystick was completely to the left, approximately 400 when the joystick was at rest, and approximately 1000 when the joystick was all the way to the left. The results were the same for the up-down position. Since the value of the integer was not completely linear with its position, we used a different ratio to calculate the color coordinates whether the joystick was to the left or right, and similarly for up and down.

Once we converted the position of the joystick to a position on the color map, we then needed to calculate the appropriate duty cycle of each color LED at any given point. We chose to do so by dividing the color map into three threshold lines like in the diagram below. The duty cycle of red, for example, would increase as the distance of the current point was below the red line and farther away from the line, by calculating the distance of the current point from this line and setting the duty cycle of red proportional to this distance from the line. The same sort of calculations were performed for blue and green. See the calculations for the duty cycle of red below as an example. Note that this calculation only applies when the "cursor" is below the line for red.

$$Distance\ from\ the\ Line = \frac{|900x + 750y|}{1171}$$

$$Red\ Duty\ Cycle = Distance\ from\ the\ Line \times \frac{100}{250}$$

These coordinates were then passed to the set_PWM() function and the duty cycle of the waveform for each color was determined by the coordinates distance from its respective side of the gamut. The further it was away, the more duty cycle increased.This was accomplished by a series of "if, else if" statements.

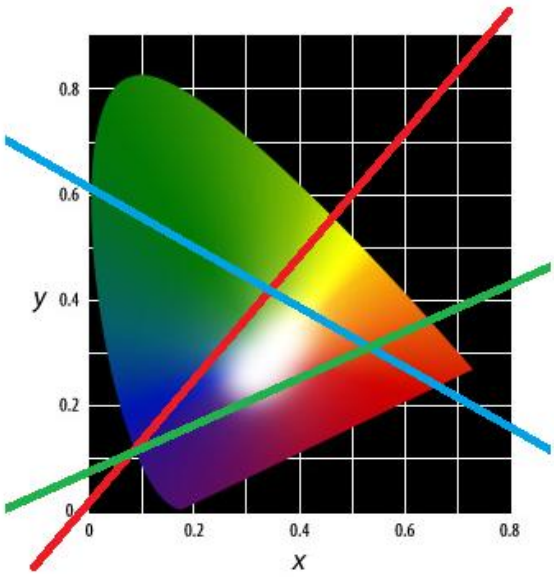

Figure 3 - 1931 Chromaticity Diagram from Reference [2]

III. Experimental Setup

A. Hex Keypad

The initial process was to just to get the key pad to be able to detect that a button was pressed and display what mode the program would be in based on the button that was pressed. Once this was verified each mode was to be integrated into the hex key pad code so it would all run from one file. If it did not run as expected some parts were commented out and some more lines of code were added to give simple output that would be seen if it was doing as expected. The process was done until some expected results were seen; by this you can sometimes debug large code structures quicker. This might only be good in such cases where the number of break points is limited, which was the case when using CodeWarrior.

B. RGB LED Strip

The strip we used has two sets of three LEDs. Each set of three LEDs for each color are run in series with a resistor. Then that is run in parallel with one other set of three RGB LEDs. In order to use the RGB LED strip we needed a way to drive them at medium to high switching speed at a voltage that is higher than what the HCS12 runs at. These LEDs are connected in a common Anode configuration (Fig. 4) and require 12v to run them.

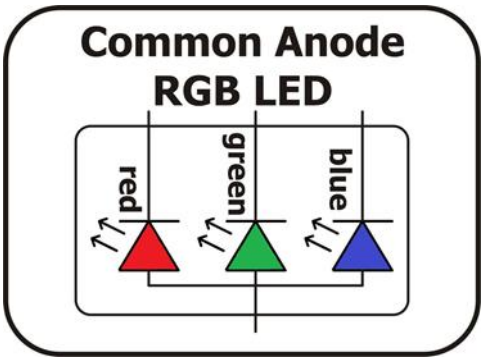

Figure 4- From Reference [4]

To drive these we decided to use an N-channel MOSFET, the one we used is BS170 from Fairchild Semiconductors [1].Considerations on selecting a MOSFET to drive the RGB LEDs are switching speed and current handling. The switching speed on the BS170 MOSFET is 10ns (as seen in figure 5), which would give about 100MHz, which is well above the 24MHz that the HCS12 will run at. The current handling of the MOSFET will have to support the current draw of each color in the RGB LED. This is a max of 100mA(each color) as seen in figure 6.

C. Mode 1 – Controlling the RGB with DIP Switches

In Mode 1, the hardware setup included the onboard DIP switches and RGB LED, and the MOSFET to RGB LED strip previously described. In this case, the RGB LED strip was driven by pins PP4, PP5 and PP6 for convenience, since the code was originally written to work with the onboard RGB LED. The RGB LED strip was added later when it became clear that the on-board RGB LED would be difficult to display to the class.

This mode implemented timer-thrown interrupts as the source of the PWM signal. Each timer regularly called its own interrupt that checked the corresponding timer counter against the "HCYCLES" and "LCYCLES" values for the color driven by that timer, and toggled the appropriate bit of Port P when the given amount of time had passed.

When the code was run, we expected to see the color displayed by the RGB change according to the position of the DIP switches in a specific way. The duty cycle of each LED was expected to increase in proportion to the binary value of each section of the DIP switch. For example, Switch 1 would be the most significant bit describing the duty cycle of the red LED, and Switch 3 would be the least significant bit. If all of the switches were off, we would expect to see no light at all, and if they were all on, we would expect to see white.

Switching Characteristics (Notes 1)

t_{on}	Turn-On Time	V_{DD} = 25V, I_D = 200mA, V_{GS} = 10V, R_{GEN} = 25Ω	BS170			10	ns
		V_{DD} = 25V, I_D = 500mA, V_{GS} = 10V, R_{GEN} = 50Ω	MMBF170			10	
t_{off}	Turn-Off Time	V_{DD} = 25V, I_D = 200mA, V_{GS} = 10V, R_{GEN} = 25Ω	BS170			10	ns
		V_{DD} = 25V, I_D = 500mA, V_{GS} = 10V, R_{GEN} = 50Ω	MMBF170			10	

Note:
1. Pulse Test: Pulse Width ≤ 300μs, Duty Cycle ≤ 2.0%.

Figure 5 - MOSFET Switching Characteristic from Reference [1]

D. Mode 2 – Temperature-Controlled Colors

Just like in Mode 1, Mode 2 controls the RGB LED strip through the MOSFETs. The only differences between the hardware setup between this mode and the previous is that this mode also uses the LCD to display the current temperature, and uses the temperature sensor as the input instead of the DIP switches.

As for the software, we accessed the temperature sensor's measurements by reading ADC pin 5. Its measurement was constantly displayed on the LCD. As a result of the coding described in the Methodology section, we expected to see the RGB LEDs display white at 24°C, fading to blue as the temperature increases with "true" (only blue LED on) blue displaying at 16C and below, and fading from white to red as the temperature increased from 24C, displaying "true" red at 43C.

E. Mode 3 – Navigating the CIE Chromaticity Diagram

This mode uses a joystick to navigate the above graph and produce the colors at each x and y coordinate. To map the x and y coordinates a joystick was wired to a breadboard and the pin for the x axis was connected to PAD 00 and the pin for the y axis was connected to PAD 01. A function, ad0conv(0), from main.asm was called to average four successive readings of the A/D and put them in intermediate variables buff0 and buff1. This method was written by Dr. Haskell and is from the Learn by Example text by him and Dr. Hanna. [3] The x and y coordinates were then determined by multiplying the coordinate by its axis length and dividing by 1024.

IV. RESULTS

A. Hex Keypad

While we tried many ways to debug the problem we were unable to resolve the underlying issue. This issue is thought to be from the Timers that the RGB and LCD use given more time, this method could have been successfully allowed us to find the problems and fix them. One thing that was noticed is that if all the parts that do anything with the LCD is commented out and mode 1 is run first then run mode 2 is run it would work. Later on it was notices that mode 1 did not work quite right but mode 2 did when using this method, but if mode 2 was run first then mode 1 it would not work at all (mode 3 what not integrated to the hex key pad code due to time constraint.

B. Mode 1 – Controlling the RGB with DIP Switches

We have found that the DIP switches did indeed control the color displayed on the RGB LEDs, and that each color was controlled by the switches we expect to control those colors, but we have found that the duty cycles applied to the LEDs did not necessarily correspond linearly with the binary value of each set of switches. In fact, the correlation between the DIP switch values and the duty cycle of the color seemed to invert or remain the same randomly.

We have determined that this result is due to the manner in which we toggled the bits of Port P. Since we used a logic command to toggle the bit, the polarity of the generated PWM would depend on the most recent value of each bit when the DIP switch value was changed. In order to fix this issue, we would recommend clearing and setting the bits according to the "HCYCLES" and "LCYCLES" count, rather than blindly toggling the bits in all cases. The reader may find a demonstration at: https://www.youtube.com/watch?v=Y76o0OshgPQ&feature=youtu.be

C. Mode 2 – Temperature-Controlled Colors

This mode ended up working exactly as expected. At room temperature of 24C, the RGB LEDs displayed white. As the temperature decreased, the color displayed faded to blue at 16C. As the temperature increased, the color displayed faded from white to red, reaching red at 43C. To improve upon this code, we would recommend increasing the range of temperature in order to smooth out the color transition and use a temperature sensor with a higher resolution. The reader may find a video demonstration at: https://www.youtube.com/watch?v=y552ksX6uTU

D. Mode 3 – Navigating the CIE Chromaticity Diagram

The waveform was then sent to LEDs. As we watched the coordinates and looked at the locations on the graph, the colors displayed on the LED strip matched the location on the chromaticity graph. Success! The reader may find a video demonstration at: https://www.youtube.com/watch?v=96VHouoOjP4&feature=youtu.be

CONCLUSIONS

We learned a lot about designing a system to run on the HCS12 microcontroller. We had many issues, while many where resolved we still had many more. Some improvements we could have made to the overall project could include: More careful programming with the timers, we could have obtained a better temperature sensor with a larger range, and optimizing the math for the color model and better hardware.

REFERENCES

[1] BS170 (PDF Datasheet N-Channel Enhancement Mode Field Effect Transistor) https://www.fairchildsemi.com/products/discretes/fets/mosfets/BS170.html?keyword=BS170

[2] "CIEXYZ - Color Models - Technical Guides." CIEXYZ - Color Models - Technical Guides. Web. 5 Dec. 2014. <http://dba.med.sc.edu/price/irf/Adobe_tg/models/ciexyz.html>.

[3] (Hanna, D., Haskell, R.; Learning by Example Using C; Rochester, MI; 2008, print)

[4] "Multi Color LEDs-Make Them In Work For Your Projects." Digital IVision Labs!:. Web. 5 Dec. 2014. <http://www.divilabs.com/2013/04/multi-color-leds-beginner-level-guide.html>.

[5] "Pushing Pixels." Pushing Pixels RSS. Web. 5 Dec. 2014. <http://www.pushing-pixels.org/2010/01/07/animations-201-color.html>.

[6] RGB 5050SMD http://www.betlux.com/product/SMD_LED/BL-LS5050A0S3.PDFBS170

■ **Absolute maximum ratings (Ta=25°C)**

Parameter	R	G	B	UR	UG	UB		Unit
Forward Current I_F	30	30	30	30	30	30		mA
Power Dissipation P_d	78	78	78	78	78	78		mW
Reverse Voltage V_R	5	5	5	5	5	5		V
Peak Forward Current I_{PF} (Duty 1/10 @1KHZ)	100	100	100	100	100	100		mA
Operation Temperature T_{OPR}	-30 to +80							°C
Storage Temperature T_{STG}	-40 to +85							°C
Lead Soldering Temperature T_{SOL}	Max.260±5°C for 3 sec Max. (1.6mm from the base of the epoxy bulb)							°C

Figure 6 - MOSFET Electrical characteristics from reference [6].