# Milestone 1: Communicating with Will Byers

*Kishan Patel—Josh Band*
Rowan University

October 17, 2017

## 1 Abstract

The overall objective for this milestone was to build an addressable RGB LED. When this RGB LED is connected to another in series, patterns could be generated. The development board that was chosen was the MSP430FR5994. This milestone is based off of the show ”Stranger Things” where a character named Will Byers tries to communicate with his mother through an alternate universe using letters on a wall and string lights. The RGB LEDs will represent the string lights in this scenario.

## 2 Introduction

Light Emitting Diodes (LED's) are seen everywhere in the world today. The LED could be used in various applications including but not limited to traffic lights, automotive headlights, traffic signals, or general lighting purposes. To be able to control what color of the LED could be very important. Today many ”Smart Home” devices are creating light bulbs that can change color by a users input on their phone. This lab is a very basic step into creating a similar device.

## 3 Design Choices

For this milestone, we had the option to choose between a variety of development boards. There was really only 1 feature that we really wanted in our boards: at least 3 PWM outputs. Out of the 5 MSP's, 3 boards were able to provide this feature: The 6989, 5529, and 5594. We could have used any of these boards. The next thing we looked at how exactly we would implement the PWM on each of the boards. On two of the three boards, we would require the use of either Timer A and Timer B, or Timer A0 and Timer A2. In the interest of keeping things simple, it was decided to go with the

5994, as that board contained at least 3 PWM pins, all on the same Timer, TimerB0. Thus, we would be able to simply set 3 of the TimerB0 Capture/Compare Registers.

## 3.1 Important Assumption

One important assumption that is made, is that the user will send perfect inputs. Meaning that the user will send a proper number of bytes every time. A ”proper” number of bytes is defined by the equation:

$$B = 2 + 3 * N \tag{1}$$

where B represents the ”proper” number of bytes, and N represents the number of Nodes or LED’s that we are configuring with this message, where N can range from 0 to infinity. As a result, we do not configure any sort of ”stop byte”, instead letting the total number of bytes define our ”stop byte”. We use a counter to count until we have processed the total number of bytes that we were told to expect in the beginning, at which point the processor will return to low power mode, to be awakened again once another byte is coming in.

## 3.2 RGB Node

Each RGB node will receive a string of Hex values over a UART RX line and will use the first four bytes received. The first byte received will represent how many bytes (including the current) are to be expected. The following three bytes will represent the Red, Green, and Blue values for the RGB LED. In order for the LED to replicate the proper color, each of the RGB values have to be converted and transformed into duty cycles. This is done for us. We are receiving a 2 bit Hex number, which can represent the same range of values as an 8-bit binary number. When receiving the hex value, the processor will automatically convert the hex value to binary if we are setting the value of a register equal to the hex value. As a result, the raw values for our duty cycles can range from 0-255. As a result, it would make sense to have the period of our PWM wave be 255. Thus, the hex value can be used directly, to give us a duty cycle ranging from 0 to 100%. After extracting the relevant duty cycle information, the Node must then pass along the rest of the RGB information via the UART TX line, on to the next node. The size byte - 3 - is passed along immediately. This is done in an attempt to maximize the throughput of data. If we can immediately pass on the number of bytes to expect, the next processor down the line can begin processing, while we are still processing our own bytes.
The RGB LED that was used was a common anode. The pin diagram for this RGB LED is shown in Figure 1. The initial schematic in Figure 2 that was created, shows a voltage source of about 3V connects to each diode color individually with a 1KOHM resister between each resistor and ground. This schematic design was causing problems. When the program ran the RGB LED initially it had a voltage supplied to it and it was lit up very dim. Since the previous schematic was giving the user issues with a false dim light from an imaginary voltage source, a new schematic was redrawn. The new schematic in Figure 3 includes MOSFET’s (N-channel). The NMOS acted as a

switch which would let the current flow if the GPIO pins went HIGH. Each resistor that went into the transistor's source was different. The reason each one was different was because a typical red LED had a voltage drop of around 2V. The green and blue LED were approximately double that of the red LED. Since there was a supply voltage of 5V each resistor would achieve a current of around 10 mA. Between the gate of the NMOS and the ground is a 1MOHM resistor which acts as a pull-down resistor so the NMOS does not stay in a HIGH state. Each gate was set to a different pin on the MSP. The gate of NMOS 1 which was connected to the red LED was connected to P3.5. The gate of NMOS 2 which was connected to the green LED was connected to P3.6. The gate of NMOS 3 which was connected to the blue LED was connected to P3.7.

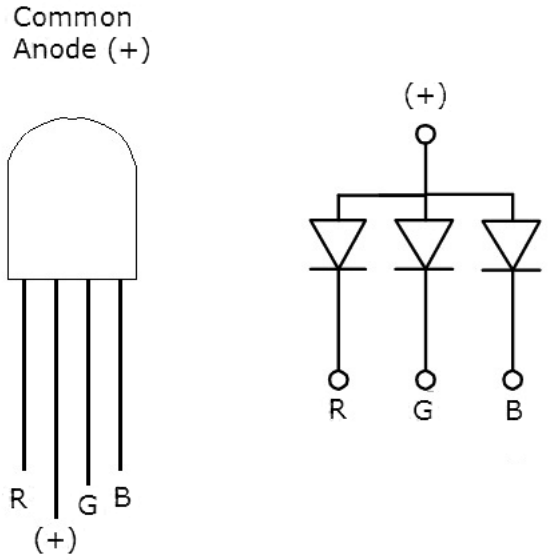


Figure 1: Common Anode RGB LED Pinout

## 3.3 UART Protocol

When communicating with the development board over UART the baud rate has to be set to 9600 baud. The derivation of equation 3.1 is as follows: each node uses 3 bytes to set their RGB value. In order to determine whether each of the first 3 bytes were removed an ending byte should be added totaling in needing at least 1 additional byte. Another byte is also added to the beginning of the message to determine the message length. Therefore, for N nodes, the number of bytes should be equation 3.1. The table below displays what the node is reading byte by byte.

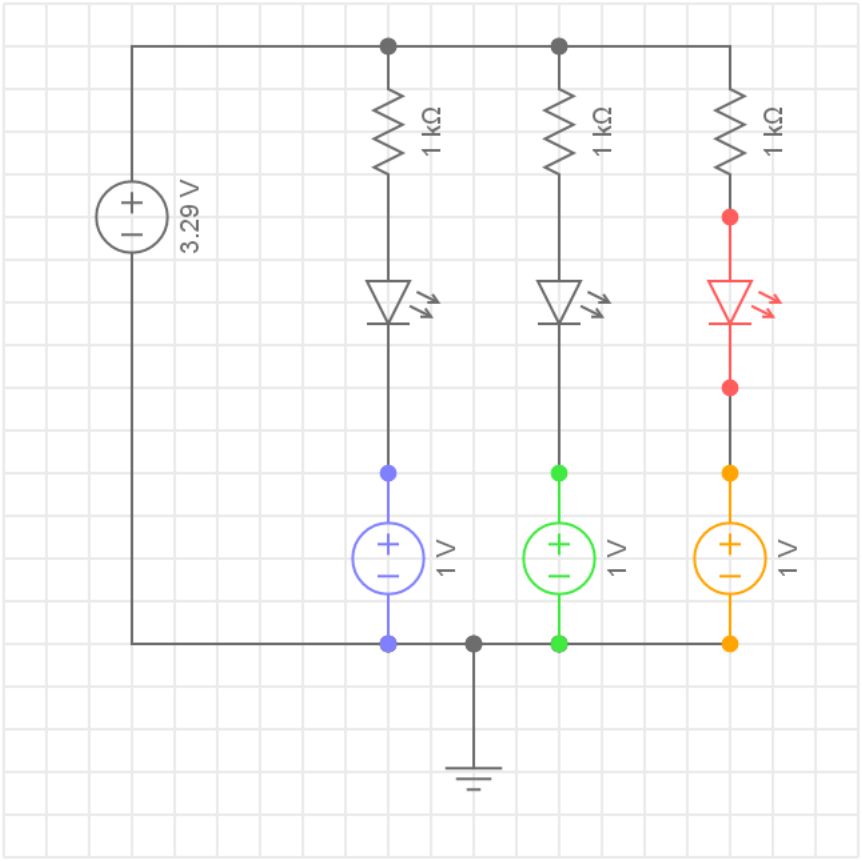


Figure 2: Beginning RGB LED Schematic

# 4 Evaluation and Results

There were two major phases that were progressed through when building this project. First, we attempted to implement the circuit shown in figure 2. Next, we improved the design of the aforementioned circuit to include MOSFET's, as shown in figure 3. One important idea to note is that both circuits work, however, they will require slightly different code in order to do so. In addition, the latter circuit worked much better than the first.

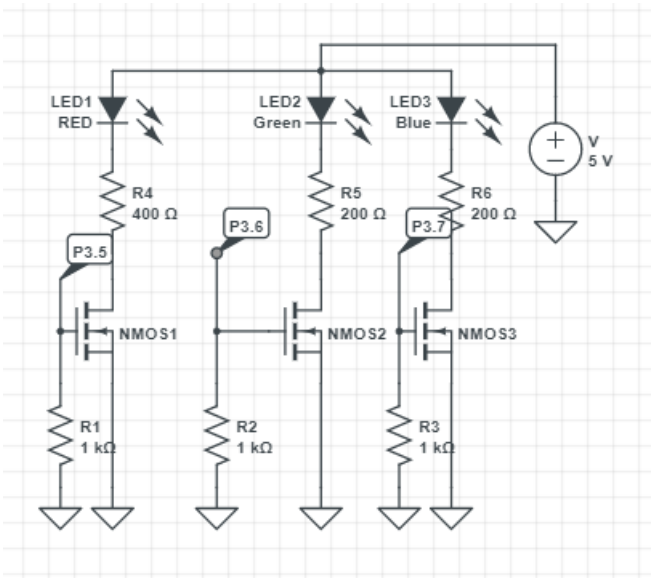


Figure 3: Final RGB LED Schematic

| Byte Number | Contents | Example |
|---|---|---|
| Byte 0 | Number of bytes (N) including this byte | 0x80 (80 bytes) |
| Bytes 1-(N-2) | RGB colors for each node | 0xFF (red) 0x00 (green) 0x88 (blue) |
| Byte N-1 | End of message character | 0x0D (carriage return) |

## 4.1 Test Cases

In order to ensure that the processor could handle any situation that we expect to be thrown at it, various situations were tested, and their performance evaluated. In order to communicate in hex over serial, a program known as realTerm is used. The cases evaluated are as follows:

- Standard Case, 5 bytes of information. Processor should take the RGB bytes, and pass along the size-3 , and the last byte.
- Data Pass through. 8 bytes of information. Processor should take its own RGB bytes, and pass along size-3, the 3 remaining RGB bytes (next nodes' bytes) and the end byte.
- Extreme Data Pass through. 14 bytes of information. Processor should do the same as above, but pass a long a larger number of bytes.
- Two Bytes. Two bytes of information. Processor should take the first byte, see that it is expecting two bytes, and then just pass along the same size byte, and the end byte, then go back into low power mode waiting for the next transmit.

# 5 Discussion/Conclusions

Overall, our code was able to handle all of our test cases exactly as we predicted. In addition, it is mentioned previously that a different circuit design is chosen. The reason this is chosen is because of the way that a common anode LED works. The innards of the RGB LED can be seen in figure 1. In order for the diodes to conduct current, the voltage values at pins R, G, and B need to be at a lower potential compared to the common anode. As a result, in order for the LED to shine brightest, it was necessary for the voltage at the pins to be equal to 0. However, the duty cycle itself is passed in via UART. Thus, in order to correct for this, for the circuit shown in figure 2, the duty cycle needed to be inverted (perform 255 - duty cycle) to get what voltage we would need to apply to the pins in order to produce the desired results. However, the aforementioned circuit yielded two issues.

- Even when powered off, the LED would still shine slightly.
- This could was not general, it could not be applied to a common cathode LED.

The first problem was not a very big issue, the shine was barely noticeable, but it was enough to be questioned. The second problem was the bigger issue. When designing a piece of software, it is generally a good idea to ensure that the software can be as versatile as possible, allowing for many additional features and uses. Inverting the duty cycle only worked for a common annode LED. As a result, the code needed to be changed so that it could be applied to both types of RGB LED's. In order to do so, some external circuitry was required. Thus, the circuit shown in figure 3 was created. This circuit employs the use of MOSFETS in order to provide a switching capability. By connecting the PWM output pins to the gates of the MOSFETS, we are able to provide the actual voltage that would be provided by the hex value duty cycle. As a result, we are able to make the code more general.

There are over 16 million $(256^3)$ combinations of colors that one simple RGB LED can make! What great way can one implement this many colors? By writing tons of code for five different MSP boards! That's how! But all jokes aside, this milestone taught many valuable lessons. Every board is unique in it own kind of way. Without performing each lab on every board one would waste a lot of time messing around with all of the timers on the other boards. The MSP430FR5994 made the code a lot simpler since it contained at least 3 PWM pins all on the same timer. LEDs are used to communicate everywhere. They are bright LEDs that practically live forever with very minimal power consumption. They are also quite affordable. Being able to control every color a RGB LED can make is making me interested in buying an LED strip in which I can program myself for my car or even my room. One can only hope to learn more about embedded systems and how powerful the MSP boards are! There are endless awesome projects in the near future!