Intro to Embedded Systems Milestone 1

Russell Binaco and Sean Hastings
Rowan University

October 17, 2017

1 Abstract

The purpose of Milestone 1 is to combine the applications learned in labs 1-4 such as UART, timers, and interrupt service routines. In this Milestone, a fictional situation is presented where many LEDs must be used in order to communicate. A UART serial connection is established between a raspberry pi and several MSP430 nodes to interface with RGB LEDs. This was implemented on an MSP430G2553 to control the RGB LED with three PWM drivers. Design factors considered included prior experience, ease of implementation, and simplicity of design. The functionality of receiving input data, controlling RGB output, and forwarding remaining data down to the rest of the network was successfully implemented and tested. An introduction, background on relevant information, evaluation, conclusion, references, and lastly an appendix of documented code are all included in this application note.

2 Introduction

In Netflix's ”Stranger Things”, Will Byers can only communicate with his mother using an array of Christmas Lights. He must be able to control which lights are on and off in order to communicate properly. Using UART serial communication, an MSP430 is used to control the values of an RGB LED. Thus, the microprocessor must be able to drive the LED with a PWM driver using the UART message protocol. Byte 0 is the total number of bytes in the message (including itself). Bytes 1 - (N-2) represent the hex value used for each RGB Node. Byte N-1 is an end of message character with a carriage return. Since the microprocessor could potentially be used at any RGB node in the line, it must be able to send the rest of the UART message to the next MSP430. Because the microprocessors will be communicating to each other, it is required that all of the UART settings such as BAUD rate are the same. In this lab a BAUD rate of 9600 is used.

3 Background

3.1 PWM

Pulse Width Modulation uses a constant-period square wave to control the percentage of time that a pin is powered high or low. At high frequencies, rapid changes in toggling an LED will appear as a brightness change to the human eye rather than the LED blinking. For this milestone, a 1KHz signal (with a period of 1ms) is used, so the ratio of time that the signal is on to the 1ms period is the duty cycle.

The MSP430G2553 can implement a hardware PWM by using its internal timers to drive a pin. The Timer A modules of the G2553 can use the capture mode of their capture/compare registers in a certain mode to drive the pin high when one CCR value is reached and drive the pin low when the other CCR value is reached. This functions concurrently with UPMODE of the Timer A modules, which reset the timer count when CCR0 is reached, to set the period using CCR0 and the duty cycle using CCR1 or CCR2. Figures 1 and 2 show the control options. OUTMOD7 and MC1 are the desired modes for the described functionality. Note: to account for a common anode configuration of the RGB LED (described in section 3.2), OUTMOD3 should be used instead.

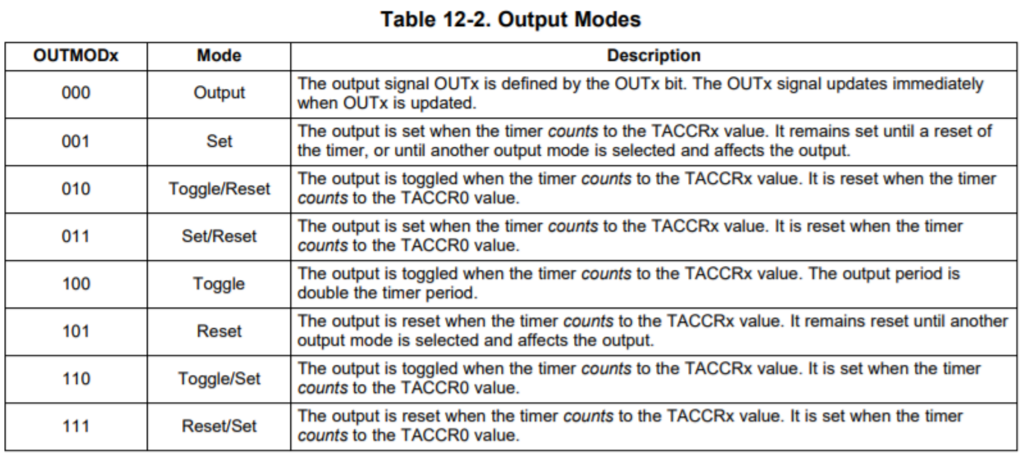

Table 12-2. Output Modes

OUTMODx	Mode	Description
000	Output	The output signal OUTx is defined by the OUTx bit. The OUTx signal updates immediately when OUTx is updated.
001	Set	The output is set when the timer *counts* to the TACCRx value. It remains set until a reset of the timer, or until another output mode is selected and affects the output.
010	Toggle/Reset	The output is toggled when the timer *counts* to the TACCRx value. It is reset when the timer *counts* to the TACCR0 value.
011	Set/Reset	The output is set when the timer *counts* to the TACCRx value. It is reset when the timer *counts* to the TACCR0 value.
100	Toggle	The output is toggled when the timer *counts* to the TACCRx value. The output period is double the timer period.
101	Reset	The output is reset when the timer *counts* to the TACCRx value. It remains reset until another output mode is selected and affects the output.
110	Toggle/Set	The output is toggled when the timer *counts* to the TACCRx value. It is set when the timer *counts* to the TACCR0 value.
111	Reset/Set	The output is reset when the timer *counts* to the TACCRx value. It is set when the timer *counts* to the TACCR0 value.

Figure 1: Table of Output Modes for the MSP430x2xx Family. {1}

By internally dividing the 1Mhz SMCLK by four, then using a CCR0 value of 256 for the period, a 977Hz frequency can be reached. Additionally, this allows the 0-255 range of RGB values to map directly to a CCR value to set the duty cycle for each component of the RGB LED that is used in this milestone. For the G2553, each Timer A only has three available CCRs, so two timers must be used to set all three RGB values since one CCR is needed to set the period of the timer. Timer A0 uses CCR1 for the red value of the LED and Timer A1 uses CCR1 and CCR2 for the green and blue values, respectively. So, once the three values for the RGB LED are received,

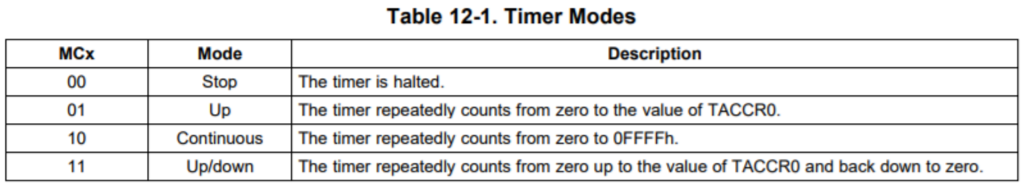

Table 12-1. Timer Modes

MCx	Mode	Description
00	Stop	The timer is halted.
01	Up	The timer repeatedly counts from zero to the value of TACCR0.
10	Continuous	The timer repeatedly counts from zero to 0FFFFh.
11	Up/down	The timer repeatedly counts from zero up to the value of TACCR0 and back down to zero.

Figure 2: Timer Mode Configurations for the MSP430x2xx Family. {1}

they can be assigned into the corresponding CCRs and the LED will change color according to the values.

3.2 RGB Node

Each RGB Node in the network is responsible for taking in a string of hex values over UART and then using the three least significant bytes to create three duty cycles. The RGB LED can use a value between 0-255, since colors are expressed by one byte. When implementing the PWM driver, it is important to consider what type of RGB LED is being used in the circuit. A common cathode LED means that each of the RGB components have a shared ground but different voltage inputs. The pinout of a common cathode RGB LED can be seen in Fig.3. A common anode, on the other hand, has a shared Vcc. This means that driving the ground pin to Vcc will turn the LED off and driving the same pin to ground will turn the LED on. To accommodate this change, the duty cycle of the PWM driver can be inverted.

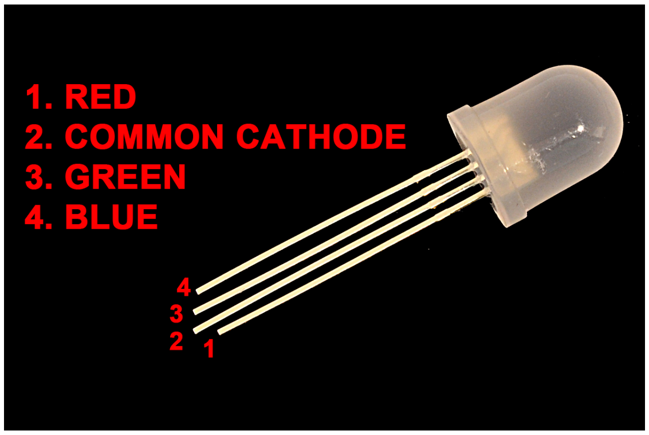

Figure 3: Pinout for a Common-Cathode RGB LED.

3.3 UART

The MSP430G2553 UART functionality is how the microprocessor can send and receive data. The microprocessor has a receive buffer and a transmit buffer. Whenever the transmit buffer is filled, the microprocessor automatically sends that out once the UART is connected correctly. When the receive buffer is filled, this triggers an interrupt that can be handled by implementing the corresponding interrupt service routine. Since the format of data that will be received, each byte of data sent to the microprocessor can be explicitly handled in the code. For example, by using a ByteCounter field, the number of bytes that have been received can be logged, and each of the first four bytes that have to be used by the microprocessor to handle the RGB values can be handled in a switch statement. Fig. 4 shows the correct connection of a UART cable to the MSP430G2553. RX, the receive pin, is P1.1, and TX, the transmit pin, is P1.2. The UART cable also needs to be connected to ground, and the power wire should not be connected.

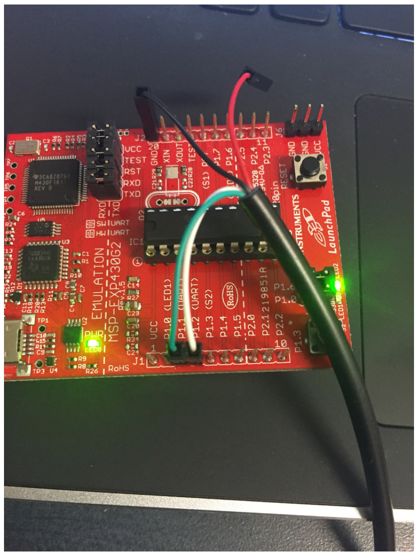

Figure 4: Picture of the UART Configuration for the MSP430G2553.

3.4 Convert-O-Box

A Convert-O-Box is simply a system that is used to take an output from one device or system and convert it into a usable form for another system. In this milestone, the Convert-O-Box will take the PWM output of the microprocessor pins, which have a current limit of 0.6mA, and convert that to the 20mA required to drive the LEDs. The Convert-O-Box used for this milestone is a high-side switch. The HSS uses a bipolar junction transistor to provide a current gain in the circuit. Fig. 5 shows the COB. The resistor values are determined using the ratio of the desired voltage drop over each resistor to the desired current through the LED. Note that the input voltage of the PWM is 3.3V and there is a 0.7V voltage drop across the BJT. Each color of LED has generally accepted values for voltage and current. Blue LEDs typically have higher voltage drops and higher current ratings than red and green LEDs.

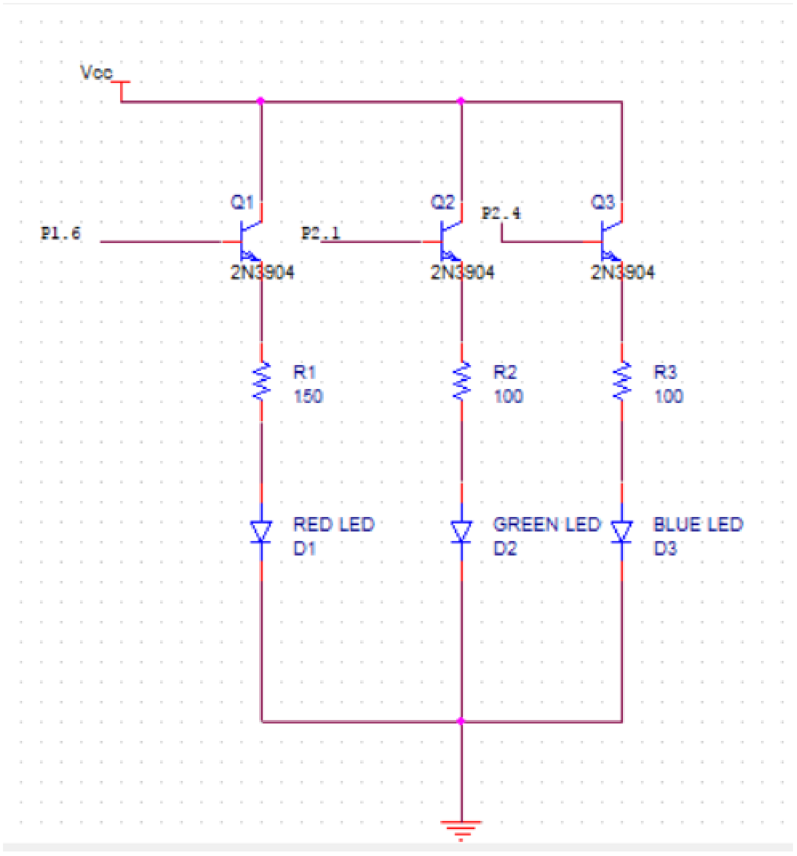

Figure 5: Schematic of PWM output with current gain to drive an RGB LED.

3.5 Equations

$$V = I * R \tag{1}$$

$$\beta = I_C / I_B \tag{2}$$

$$I_C + I_B = I_E \tag{3}$$

These equations are relevant to the Convert-O-Box used to transfer the PWM output of the microprocessor to power the RGB LED.

4 Evaluation

The main design decision involved in this milestone was the decision of which of the 5 microprocessors used in this course should be used. First, the MSP430FR5994 was used. This microprocessor has a Timer B which has between four and seven capture/compare registers, which would eliminate the need for two Timer A modules running. However, the FR family of microprocessors has a more complicated implementation of UART interrupt service routines (there is only one ISR for all of the transmitting and receiving interrupts). Also, the UART pins used for the MSP430G2553 are easily accessible on the board, while the FR family boards have two RX pins and two TX pins that are normally covered. While using sample code from Code Composer Studio, testing UART functionality on the FR5994 was not working. Russell had already implemented UART on the G2553 for the lab 2 extra work, and tested it using the correct COM port on Putty, so getting UART to work as necessary for the milestone was much easier on the G2553. The changes to the code for using two timers rather than one were minimal, and there is no loss of functionality as a result of those changes.

For the Convert-O-Box, there is clearly more power dissipation in a BJT rather than a MOSFET. The circuit was first attempted using NMOS transistors rather than NPN BJTs, and the circuit failed to turn on the LED. This is most likely due to an insufficient voltage or current requirement across the LEDs as a result of the circuit. As discussed in class, a two-stage MOSFET COB may be needed, so the successful BJT circuit reduced the number of components used in the circuit overall.

One other design decision pertains to the exact accuracy of transferring RGB values from UART into PWM signals. In order to have a 1:1 relation from input to CCR value, the internal SMCLK was divided by 4 and the CCR0 value for each timer was set to 256. This results in a frequency of 977Hz, which is slightly slower than the 1KHz that was defined in the assignment. However, the other possible implementation would be multiplying the RGB value by 4 to get a range of 0 to 1020. This would mean that RGB values of 250-255 would all result in a 100 percent duty cycle. It was determined that a small change in frequency is better than a small loss in accuracy, so the former option was used.

One of the most common issues I have seen in the past is the separation between the Results and the Discussion within a app note. Most students tend to mix these two portions of their reports into one big section. In an app note or a technical paper, you want to separate out your data (results) into one section and your discussions into another. This way, you can easily reference back to the results as well as it provides one place where a reader can see all of your data.

5 Results and Discussion

Fig. 7 shows the completed circuit implemented on a breadboard with the microprocessor. The breadboard contains the RGB LED and the COB, and the microprocessor provides the breadboard with 3.3V for VCC and a ground. The microprocessor is also connected to UART. When testing the implementation by using Putty to provide input to the UART, the LED changed as expected due to each update of a CCR value. This testing was limited to providing the ASCII conversion of keyboard input, whereas the in-lab milestone demo will be receiving values from 0 to 255. However, by initializing the CCR values to two zeros and one 255, the LED was determined to successfully show all three colors with correct CCR settings. Furthermore, Putty received the fifth and later bytes successfully after the first four were read in, and stopped receiving more bytes after the number specified by the first byte sent to the microprocessor. Fig. 6 shows the LED successfully lit after an update to each of the RGB CCR values. The only potential under-performance of the implementation is its failure to assign CCR values using a log scale. Since humans see in a log scale, linearly increasing CCR values with input RGB values may not cause the correct colors to be shown. However, if RGB values take into account this logarithmic vision, no change would be necessary.

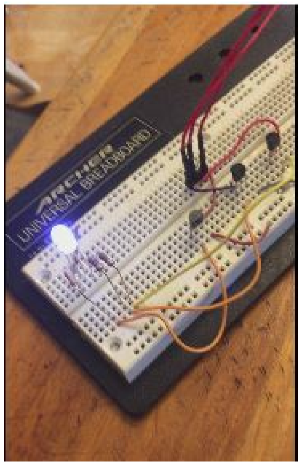

Figure 6: A blue output of the RGB LED.

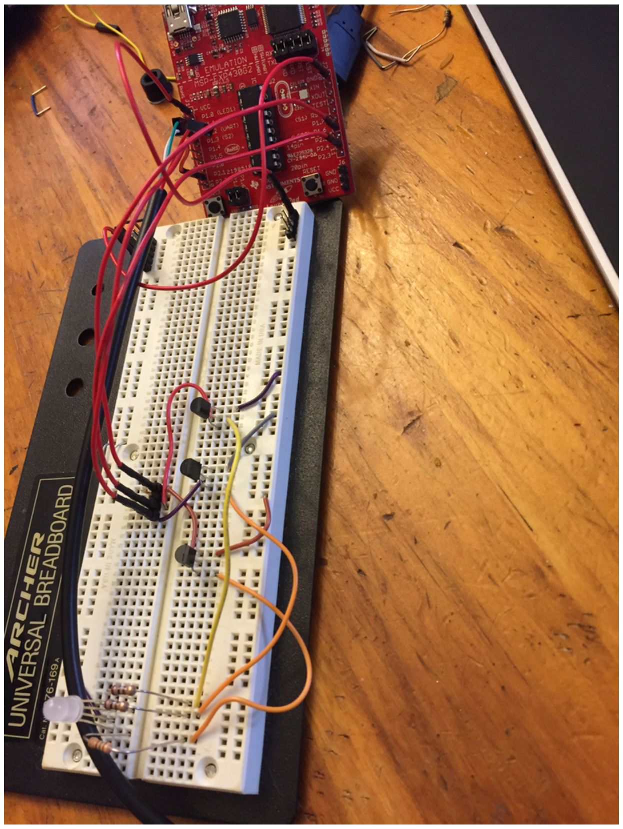

Figure 7: Complete circuit of the MSP430G2553 PWM driver with RGB LED output.

6 Conclusion

The purpose of this milestone was to apply all of the skills learned in the first four labs to accomplish a meaningful task. By using UART, timers, and interrupt service routines, the core aspects of the first four labs are reflected in this milestone. The LED communication in the show Stranger Things is a fictional situation, but it provides a concept that is easily applied to microprocessors that is also an interesting use of PWM with LEDs.

7 References

1. Family User guide for MSP430x2xx: http://www.ti.com/lit/ug/slau144j/slau144j.pdf

8 Appendix

```
#include <msp430.h>
int numOfBytes = 0;
int byteCount = 0;
int temp = 0;
int main(void)
{
    WDTCTL = WDTPW — WDTHOLD; // stop watchdog timer
    //Timers Config
    P1DIR —= BIT6 + BIT1; // P1.6 output
    P1SEL —= BIT6; // P1.6 for TA0 CCR1 Output Capture
    P1SEL2 = 0x00; // Select default function for P1.6
    TA0CCR0 = 256; // PWM Freq=1000Hz
    TA0CCTL1 = OUTMOD_7; // CCR1 reset/set: set on at CCR0, off at CCR1 capture
                                //(see table 12-2 in specific datasheet)
    TA0CCR1 = 255; // CCR1 duty cycle
    TA0CTL = TASSEL_2 + MC_1 + ID_2; // SMCLK/4, up mode, 1MhZ
    P2DIR —= (BIT1 — BIT4); // P2.1,4 output
    P2SEL —= (BIT1 — BIT4); // P2.1,4 for TA0 CCR1 Output Capture
    P2SEL2 = 0x00; // Select default function
    TA1CCR0 = 256; // PWM Freq=1000Hz
    TA1CCTL1 = OUTMOD_7; // CCR1 reset/set: set on at CCR0, off at CCR1 capture
                                //(see table 12-2 in specific datasheet)
    TA1CCTL2 = OUTMOD_7;
    TA1CCR1 = 0; // CCR1 duty cycle
    TA1CCR2 = 0;
    TA1CTL = TASSEL_2 + MC_1 + ID_2; // SMCLK/4, up mode, 1MhZ
    //UART Config
    if (CALBC1_1MHZ==0xFF) // If calibration constant erased
    {
        while(1); // do not load, trap CPU!!
    }
    DCOCTL = 0; // Select lowest DCOx and MODx settings
    BCSCTL1 = CALBC1_1MHZ; // Set DCO
    DCOCTL = CALDCO_1MHZ;
    P1SEL —= BIT1 + BIT2 + BIT5; // P1.1 = RXD, P1.2=TXD
    P1SEL2 = BIT1 + BIT2 ; // P1.1 = RXD, P1.2=TXD
    UCA0CTL1 —= UCSSEL_2; // SMCLK
    UCA0BR0 = 104; // 1MHz 9600
    UCA0BR1 = 0; // 1MHz 9600
    UCA0MCTL = UCBRS0; // Modulation UCBRSx = 1
    UCA0CTL1 &= ˜UCSWRST; // **Initialize USCI state machine**
    IE2 —= UCA0RXIE; // Enable USCI_A0 RX interrupt
    __bis_SR_register(GIE); // Enter LPM0, interrupts enabled
    while(1)
    {
    }
}
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)
{
    while (!(IFG2&UCA0TXIFG)); // USCI_A0 TX buffer ready?
temp = UCA0RXBUF;
switch(byteCount){
    case 0:
        numOfBytes = temp; //first byte input
        break;
    case 1:
        TA0CCR1 = temp; //Red value
        break;
    case 2:
        TA1CCR1 = temp; //Green value
        break;
    case 3:
        TA1CCR2 = temp; //Blue value
        UCA0TXBUF = numOfBytes-3; //send updated numBytes
        break;
    default:
        if(byteCount<numOfBytes)
            UCA0TXBUF = temp; //send remaining bytes
        }
    }
    byteCount++;
}
```