Milestone 1: UART Controlled RGB LED

Nick W, John G, and Dan M
Rowan University

October 22, 2018

1 Abstract

The MSP430 microcontroller can be programmed for serial communication utilizing UART (Universal Asynchronous Receiver/Transmitter) circuitry. Serial communication was established with a computer by using the Realterm software to send packets of information directly to the MSP4305529LP board through USB TX/RX connection. The general purpose of this milestone is to control the intensity and color of an RGB LED utilizing serial communication singularly or in a nodal network.

2 Introduction

The practical purpose of a microcontroller is to provide control for various mechanical and electrical systems through the use of digital and analog signals.

Milestone 1 required us to perform the following tasks utilizing the MSP430 microcontroller:

- Control three different LED colors utilizing either hardware or software PWM (Pulse Width Modulation).
- Receive data using serial communication from terminal software.
- Send and receive data using the RX/TX pins available on the microcontroller to other microcontroller boards to create a node system.

In order to satisfy each requirement listed above, it is critical to understand core concepts such as serial communication, microcontroller functionality, and RGB LED characteristics.

3 Background

3.1 Pulse Width Modulation

Pulse Width Modulation, or also known as PWM, is a type of digital signal that is derived by the average value between the time a signal is high (a value greater than 1) and the time a signal is low (a value generally equal to 0). The result of this relationship is often defined as a signal's duty cycle, which is defined in Equation 1 below:

$$\text{Duty Cycle} = \frac{T_{on}}{T_{on} + T_{off}} \times 100\%$$

T_{on} = Time the signal is high
T_{off} = Time the signal is low

PWM can be utilized to set the intensity of light as well as the color by altering the value of T_{on}. When the value of T_{on} approaches the value of T_{total}, the duty cycle increases. This increase in duty cycle will cause the LED to appear brighter, and will reach maximum brightness when the duty cycle approaches 100%.

3.2 MSP430F5529LP Microcontroller

For the purpose of this milestone, the MSP4305529LP microcontroller was selected. Below are a key few features of the board:

- Four 16-bit timers
- 12-bit analog-to-digital converter (ADC)
- Hardware multiplier
- 63 I/O pins.

The primary reason why the MSP430F5529LP board was selected was due to its access to seven capture and compare registers available on a single timer (TA0CCR0 - TA0CCR6). This number of capture and compare registers enabled our team to utilize software PWM to achieve RGB LED control.

3.3 The RGB LED

Essentially, an RGB LED is a red, green, and blue LED in one package. They come in two variations, common cathode and common anode. For the milestone, a common anode was used (1). This means that the red, green, and blue elements share a common ground. In order to power one of them, 3.3V or 5V is supplied from the MSP4305529's pins.

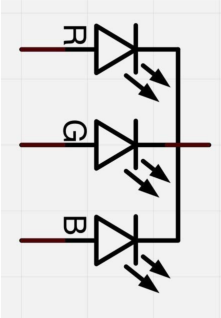

Figure 1: Common Anode RGB Schematic

By controlling the brightness of the individual colors, a desired color can be created; much like how an artist mixes colors on a palette (2). In order to control the brightness of the LEDs, PWM is used. In the case of the Milestone, the brightness can be any value from 0 (Off) to 255 (Max brightness), these values set the duty cycle of each PWM signal.

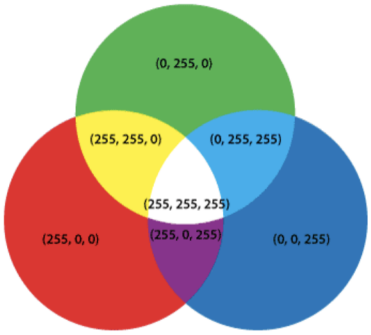

Figure 2: A color wheel showcasing the possible combinations of the RGB LED

The RGB was diffused, this means that instead of being clear, the casing was opaque. This results in a cleaner mixing of colors during operation from close up. When tested with a multi-meter, the RGB used 3mA per color at max brightness and 3.3V. Since this required less current to operate than a conventional RGB, no additional high or low side switches were needed.

3.4 Serial Communication

Serial communication is one of two communication protocols. A communication protocol is a system or set of rules that allows two entities to transmit data. The other communication protocol is parallel communication. The main difference between these

to is that parallel communication transmits multiple pieces of data at a time through parallel wires, while serial communications can only send data bit by a bit usually through two wires one for transferring data and one for receiving data. In this lab serial communications were utilized.

3.4.1 Asynchronous VS. Synchronous

There are two types of serial communication transmissions synchronous and asynchronous. Asynchronous transmissions can be sent at any time and are separated by a stop bit and a start bit. These stop bits and start bits ensure that data transmissions are received correctly. Synchronous transmissions don't' need stop bits or start bits, because the data is transmitted at a clock frequency that is synchronized between the transmitted devices. Synchronous transmissions are often more expensive, but data can be transmitted faster using this method. In this lab asynchronous transmissions were utilized since it does not depend on synchronizing multiple devices.

3.4.2 UART

UART stands for universal asynchronous receiver-transmitter. UART is actually a hardware device that is used in asynchronous serial communication. UART is a device that takes data in the form of bytes and sequentially decomposes the data into individual bits so that it can be transmitted serially. At the destination these bit are re-assembled into the original data component. UART is the most common serial communication device for microcontrollers, in fact most MSP microcontrollers utilize the UART protocol, including the MSP4305529LP.

4 PWM Generation

The first task of the Milestone required the control of an RGB LED utilizing PWM. To control the output of a standard LED, it is required for a timer to continuously count up to a specified value and trigger an interrupt once that specified value is reached. In the case of controlling an RGB LED, the same principle would apply but would instead require three separate interrupts; one for red, green, and blue respectively.

Initially, the timer used to continuously count was selected to be SMCLK due to its high frequency. The timer was additionally set to Up Mode, such that an interrupt would trigger once the value in the TA0CCR0 register was met by the timer register (TA0R). The value in the TA0CCR0 capture and compare register was set to be 256 to limit the period of the entire signal.

In order to set the intensity of red, green, and blue, each color was designated by a specific capture and compare register.
Furthermore, red, green, and blue were assigned to pins 1.3, 2.0, and 2.3 of the MSP4305529LP, respectively.

Capture/Compare Register	Functionality	Initial Value
TA0CCR0	Sets the maximum of the signal. Will reset once value is met.	256
TA0CCR1	Register designated for the Red LED	0
TA0CCR2	Register designated for the Red LED	0
TA0CCR3	Register designated for the Red LED	0

As seen in the table above, TA0CCR1 was designated to red, TA0CCR2 was designated to green, and TA0CCR3 was designated to blue. In order to generate a PWM signal for each color, TA0CCR1, TA0CCR2, and TA0CCR3 requires a unique interrupt that occurs once the value in each capture and compare register is equal to TA0R. To achieve this, the TA0IV interrupt vector generator was used. In short, the TA0IV interrupt vector generator can be used to determine which flag requested an interrupt when multiple interrupts exist. The flag is ultimatly determined by priority, with TA0CCR0 holding the highest priority and TA0CCR6 holding the least priority. By utilizing a switch-case statement with TAOIV, each capture and compare register was assigned a specific interrupt utilizing the table below in Figure 1.

Bit	Field	Type	Reset	Description
15-0	TAIV	R	0h	Timer_A interrupt vector value 00h = No interrupt pending 02h = Interrupt Source: Capture/compare 1; Interrupt Flag: TAxCCR1 CCIFG; Interrupt Priority: Highest 04h = Interrupt Source: Capture/compare 2; Interrupt Flag: TAxCCR2 CCIFG 06h = Interrupt Source: Capture/compare 3; Interrupt Flag: TAxCCR3 CCIFG 08h = Interrupt Source: Capture/compare 4; Interrupt Flag: TAxCCR4 CCIFG 0Ah = Interrupt Source: Capture/compare 5; Interrupt Flag: TAxCCR5 CCIFG 0Ch = Interrupt Source: Capture/compare 6; Interrupt Flag: TAxCCR6 CCIFG 0Eh = Interrupt Source: Timer overflow; Interrupt Flag: TAxCTL TAIFG; Interrupt Priority: Lowest

Figure 3: TA0IV Cases

Based on the internal properties of the MSP430F5529LP, this approach must be utilize to instantiate multiple interrupts on a single timer.

To simulate the PWM signal, it is determined that the output of each pin used would be set low once the value in TA0CCR1, TA0CCR2, and TA0CCR3 were equal to the value in TA0R. The output of each pin would then be set high once the value in TA0CCR0 register was equal to the value in TA0R. This low-to-high behavior would replicate the behavior seen in Reset/Set Output Mode seen in Figure 2.

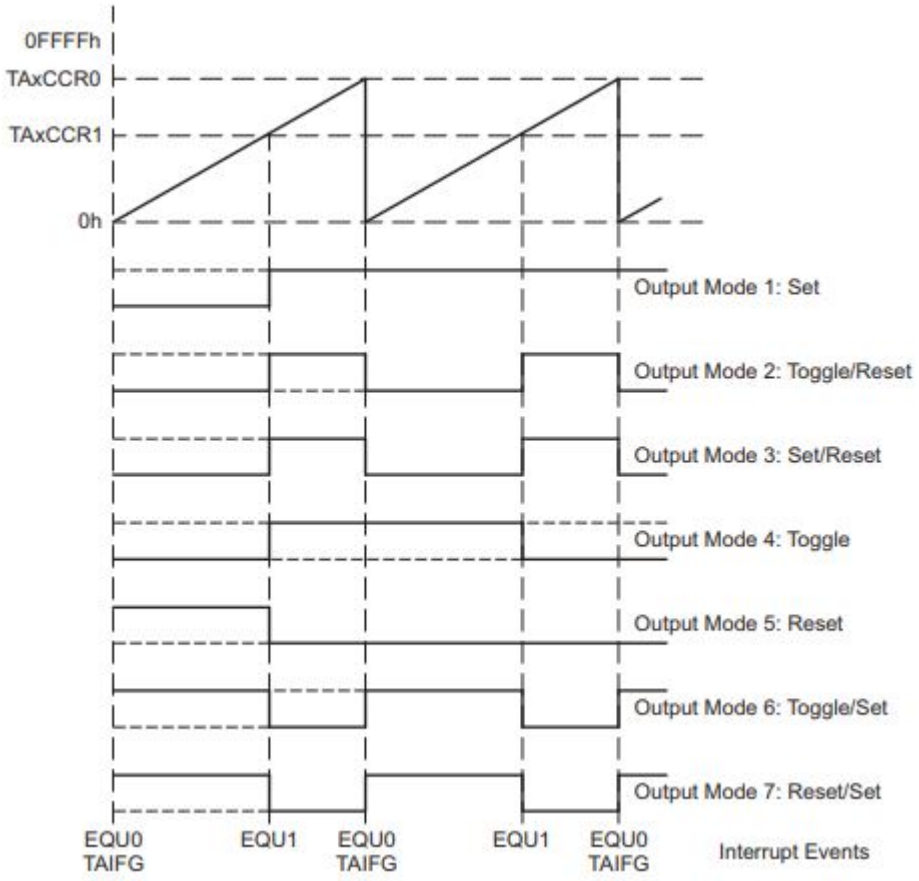

Figure 4: MSP430 Output Modes

5 UART Application

UART was utilized as an interrupt for the microcontrollers processor. The receive buffer and transmit buffer were then used in this interrupt to control the intensity of each of the RGB LEDs. This was done by using values received in the Rx buffer and saving them in the CCR register correlating to that RGB's color. This was implemented using a switch case inside the UART interrupt. Two variables were defined at the top of the program these values were ”size” and ”bit”. The ”size” variable was simply the size of the packet that was being received from the Rx buffer and the ”bit” variable was a counter for which byte of the packet message was being received. The ”bit” variable is also what the variable being checked by the switch statement. The ”bit” variable gets incremented at the end of each interrupt and once the ”bit” variable is equal to the ”size” variable the ”bit” variable is reset to 0. The zero case for the switch

case was simply to load the first value received into the ”size” variable and then send the value received minus three to the TX buffer. The next case, case one, was used to set the intensity of the red LED. This is were the incoming byte is saved into the capture control register correlating the certain color, in this case the CCR1 register. The second case sets the green LED intensity using the CCR2 register and the third case sets the blue LED intensity using the CCR3 register. After the ”bit” is greater than three the switch case goes to the default case which simply passes the byte in the Rx buffer into the TX buffer until the entire message is processed. Once the message is processed the UART interrupt is essentially reset and ready to receive another message to be processed.

6 Discussion and Results

6.1 Replicating colors via UART

The Milestone test consisted of a color check via UART. The MSP4305529LP was connected to the serial data port of the test computer, and the transmission speed was set to 9600 baud. The test colors were sent in their respective bits:

1. Red (0x04 0xFF 0x00 0x00)
   This was to test if a pure red could be replicated.
2. Green (0x04 0x00 0xFF 0x00)
   This was to test if a pure green could be replicated.
3. Blue (0x04 0x00 0x00 0xFF)
   This was to test if a pure blue could be replicated.
4. White (0x04 0xFF 0xFF 0xFF)
   This was to test if a pure red green and blue could be replicated, creating white.
5. 1/2 White (0x04 0x7E 0x7E 0x7E)
   This was to test if the program and board could handle half duty cycle PWM. 0x7E is half of 0xFF, setting the brightness of all the pins to half.
6. String of bytes (Figure 5)
   This was to test the program’s abilities to receive data, decode it, and forward the data.

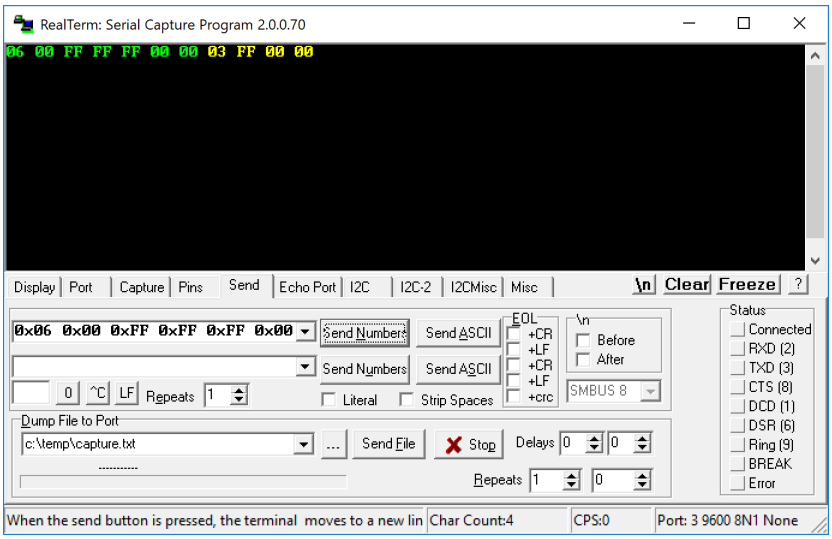

Figure 5: RealTerm string of bytes

Figure 5 illustrates the RealTerm functionality of the MSP4305529LP in the 'string of bytes' test. The half-duplex option was selected in RealTerm, allowing for the MSP4305529LP's output to be seen (Yellow). As seen, it receives the bits (in green), takes the three it needs, then subtracts the package length and forwards the rest (in yellow).

The three bits the MSP4305529LP receives are: 0x00 0xFF 0xFF. This produces a teal color (Figure 6). The bytes it uses are then subtracted from the package and the rest of the data is forwarded. Had it been in a chain during this test, the next MSP430 in line would produce a red color using the bytes 0xFF, 0x00, 0x00. In this case, the data was sent back to RealTerm.

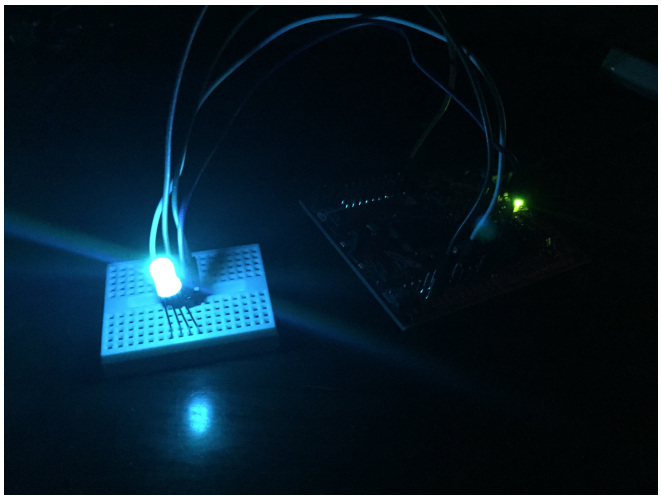

Figure 6: Teal color from final test, taken in dark room to see color more clearly

6.2 The Chain

The final task of the Milestone was to chain multiple MSP430 boards together and control the individual colors of each utilizing serial communication. In order to achieve this, RealTerm was used to send a package of bits to the first in the chain. The other MSP430 boards were connected using their RX and TX pins. The package contained the package length and the colors desired stored in values ranging from 0 to 255. When the first MSP430 board received the package, it decoded the information it needed and then decremented the package length accordingly by a value of three, sending the rest of the data to the next MSP430 board.

As seen in figure 7, the chain was sent: RED (0xFF, 0, 0), GREEN (0, 0xFF, 0), BLUE (0, 0, 0xFF), and VIOLET (0xFF, 0, 0xFF).

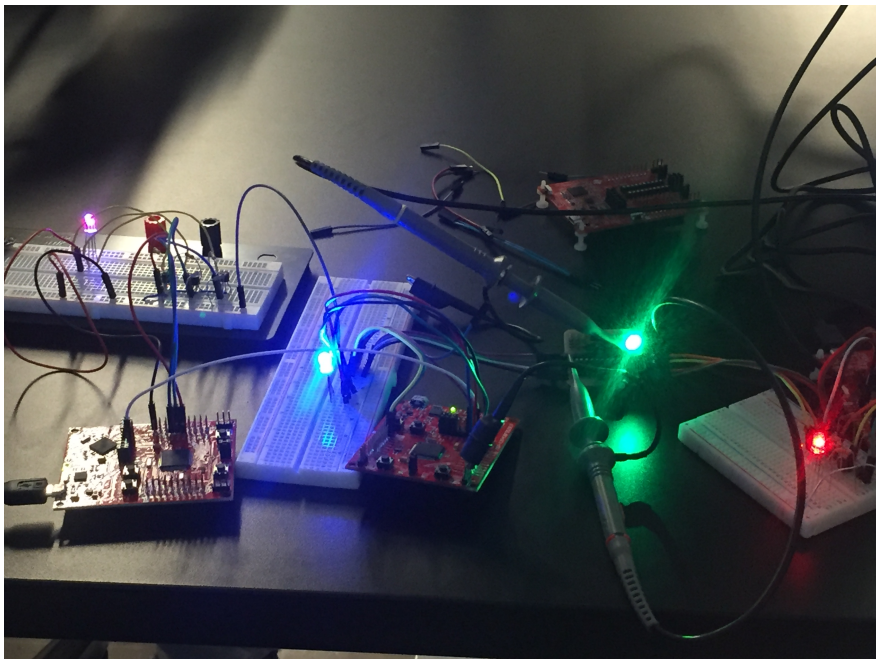

Figure 7: Two MSP430G2553 and two MSP430F5529 boards chained together

6.3 Issues and resolutions

6.3.1 Hardware issues

Initially, the MSP430G2552 was used. In order to use Up Mode (MC_1), a register was needed to set the limit that the clock counts up to. Three additional registers were needed for red, green, and blue. The board ultimately chosen for these tasks was the MSP430F5529 because it allowed enough capture and compare registers on a single timer to achieve multiple timer interrupts.

In another implementation of the milestone, the MSP430G2553 could potentially be used. Such implementations could be a hardware PWM or a software PWM using continuous mode. Another method of achieving this task using the MSP430G2553 includes utilizing two separate timers to achieve the desired number of capture and compare registers.

6.3.2 Software issues

One issue that arose when setting the duty cycle of the PWM signals occurred when the board was sent all zeros. This process involved setting the values in the TA0CCR1, TA0CCR2, and TA0CCR3 registers to 0, which would theoretically set the duty cycle to zero and shut the LEDs off. However, setting these values caused each LED to underflow and output the highest possible value, causing the LED to become white.

In order to fix this, the I/O on the color pins was halted if a 0 was received. Once a value besides 0 was read by the program, each pin would have I/O functionality restored. By completely turning off the I/O function, the RGB could not possibly turn on and provided the desired results.

Additionally, it was observed that the RGB would tend to flicker when the board was reset. In order to remedy this issue, each color pin had their I/O functionality disabled initially. In order to do this, the individual color registers (TA0CCR1, TA0CCR2,TA0CCR3) were to set to 0. Once a value besides 0 was read by the program, these outputs were restored by the remedy discussed above.

7 Conclusion

The MSP430F5529 was programmed to address the pins of an RGB to create a desired color. In order to send the desired colors to it, RealTerm was used via serial communication. Additionally, it was used in a chain with other MSP430s to create color patterns. When a package of bytes was received via UART, the MSP successfully:

1. Received the bytes
2. Subtracted three from the package length
3. Used the three bytes to set the RGB's color
4. Forwarded the package to the next microcontroller in the string or to RealTerm

This lab presented a myriad of new information necessary for embedded software design. This lab amalgamated all the topics previous learned into a single program. This code utilized PWM, interrupts, UART, and even a few necessary hardware devices. Overall this lab was good practice in utilzing as many embedded skill as possible to create complex and effective programs.

8 Appendices

```
#include <msp430.h>

volatile int bit = 0, size = 0;      // Initialize and type define variables
                                     // for later use

int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;      // Stop WDT

/* Port Configuration */
  P1DIR |= BIT3;              // Sets Pin 1.3 as output
  P2DIR |= BIT3;              // Sets Pin 2.3 as output
  P2DIR |= BIT0;              // Sets Pin 2.0 as output
  P2OUT &= ~BIT0;             //Sets Pin 2.0 as 0 initially
  P1OUT &= ~BIT3;             //Sets Pin 1.3 as 0 initially
  P2OUT &= ~BIT3;             //Sets Pin 2.3 as 0 initially
  P3SEL |= BIT4;              //Enables UART connection
  P3SEL |= BIT3;              //Enables UART connection
  P4SEL |= BIT4 + BIT5;       //Enables UART connection

/* Clock Configuration */
  UCA1CTL1 |= UCSWRST;
  UCA1CTL1 |= UCSSEL_2;       // SMCLK
  UCA1BR0 = 6;                // 1MHz BAUD Rate = 9600
  UCA1BR1 = 0;                // 1MHz, BAUD Rate = 9600
  UCA1MCTL |= UCBRS_0;        // Modulation
  UCA1MCTL |= UCBRF_13;       // Modulation
  UCA1MCTL |= UCOS16;         // Modulation
  UCA1CTL1 &= ~UCSWRST;       // **Initialize USCI state machine**
  UCA1IE |= UCRXIE;


/* Capture & Control Configuration */
  TA0CCTL1 = CCIE;
  TA0CCTL2 = CCIE;
  TA0CCTL3 = CCIE;
  TA0CTL = TASSEL_1 + MC_1 + ID_0 + TAIE;    //SMCLK, UPMODE, NO DIVIDER, CLEAR
     ,

/* Initialize Registers to 0 */
  TA0CCR1 = 0;      //initialized Red LED to be off
  TA0CCR2 = 0;      //initialized Green LED to be off
  TA0CCR3 = 0;      //initialized Blue LED to be off
  TA0CCR0 = 255;    //initialized max PWM length


  __bis_SR_register(GIE);    // Enter LPM0, interrupts enabled
  while(1) {
      if (TA0CCR1 == 0)      //Checks to see if the Red LED should be on. If
    the value is 0, then the LED will be off.
      {
          P1DIR &= ~BIT3;    // Turns the LED pin off as long as the CCR
                             // register is at 0
      }
```

```
53          else {
54              P1DIR |= BIT3;     // If the CCR1 is anything but 0, LED turns on
55          }
56          if (TA0CCR2 == 0)      // Checks to see if the Green LED should be on
57          {
58                P2DIR &= ~BIT0; // Turns the LED pin off as long as the CCR
59                                // register is at 0
60          }
61          else {
62                 P2DIR |= BIT0; // If the CCR2 is anything but 0, LED turns on
63          }
64          if (TA0CCR3 == 0)  //Checks to see if the Green LED should be on
65          {
66              P2DIR &= ~BIT3;  // Turns the LED pin off as long as the CCR
67                               // register is at 0
68          }
69          else {
70              P2DIR |= BIT3;   // If the CCR3 is anything but 0, LED turns on
71          }
72      }
73  }
74
75  #pragma vector=TIMER0_A1_VECTOR
76  __interrupt void Timer0_A1_ISR (void)   //CONTROLS PWM
77  {
78      switch(TA0IV)
79      {
80      case 2: //Case 2 defines interrupt on CCR1
81          P1OUT &= ~BIT3;      // Turn Red LED off for CCR1 interrupt
82          break;
83
84      case 4: //Case 4 defines interrupt on CCR2
85          P2OUT &= ~BIT0;      // Turn Green LED off for CCR1 interrupt
86          break;
87
88      case 6: //Case 6 defines interrupt on CCR3
89          P2OUT &= ~BIT3;      // Turn Blue LED off for CCR1 interrupt
90          break;
91
92      case 14:
93          P1OUT |= BIT3;       // Set Red LED
94          P2OUT |= (BIT0 | BIT3);  // Set Green and Blue LED
95          break;
96      }
97  }
98  #pragma vector=USCI_A1_VECTOR           // Interrupt vector definition
99  __interrupt void USCI_A1_ISR(void) {    // Interrupt function declaration
100
101     switch(bit) {                       // Switch statement for byte position
102         case 0 :                        // Sets Length Byte
103         while(!(UCA1IFG & UCTXIFG));    // Checks to make sure TX Buffer
104                                         // is ready
105         size = UCA1RXBUF;
106         UCA1TXBUF = UCA1RXBUF - 3;      // Sends length byte to the
107                                         // Transfer Buffer
108         break;
109         case 1 :                        // Red LED Set
```

```
110             TA0CCR1 = (UCA1RXBUF);          // Sets CCR1 value to incoming byte
111             break;
112         case 2 :                            // Green LED Set
113             TA0CCR2 = (UCA1RXBUF);          // Sets CCR2 value to incoming byte
114             break;
115         case 3 :                            // Blue LED Set
116             TA0CCR3 = (UCA1RXBUF);          // Sets CCR3 value to incoming byte
117             break;
118         default:                            // For all of the other bytes
119             while(!(UCA1IFG & UCTXIFG));// Checks to make sure TX buffer
120                                             // is ready
121             UCA1TXBUF = UCA1RXBUF;          // Passes incoming byte from receive
122                                             // buffer to outgoing byte through
123                                             // the transfer buffer
124         break;
125     }
126
127     if(bit != size ){               // If the value for bit is not equal to the
128                                     // size variable
129         bit ++;                     // Increment the bit value
130     }
131     else {                          // If the value for bit is equal to size then
132         bit = 0;                    // reset the bit value
133     }
134 }
```