Milestone 1 Application Note: Stranger Things Light Wall

Rowan University
By: Timothy Gordon, Ryan Hare, and Jessica Wozniak

1. Overview

In the Netflix series “Stranger Things,” Will Byers becomes trapped in a parallel universe. The only way he can communicate with his mom is to make Christmas lights light up. His mom then hangs up Christmas lights like a Ouija board on the wall. Based on this scene this program was designed to be a single node in a chain of nodes, being controlled by a master. Communication for the program is done over UART. Each node is required to take in a string of HEX values. Byte 1 is the total number of bytes in the package and bytes 2, 3, and 4 are translated into different duty cycles to make different colors on the RGB LED. After the node translates the bytes to duty cycle, the remainder of bytes are repackaged and sent to the next node.

2. RBG LEDs

There are two different types of RGB LEDs: common anode and common cathode. In a common anode RGB one anode is used by all three color LEDs and in a common cathode, one cathode is used by all three LEDs (see *Figure 1*).

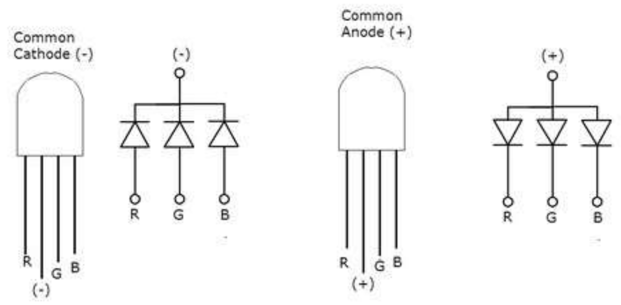


Figure 1: RGB LED

A common anode RGB LED was used in this application.

3. The circuit

The circuit to run the program was composed of BJTs, resistors and a RGB LED. On each BJT the emitter was connected to ground. A 2.4 kΩ resistor (Resistor B) was connect between the base of the BJT and the pins on the MSP430FR5994, and a 100 Ω resistor (Resistor C) was connect between the collector of the BJT and the corresponding RGB LED pins. The anode of the RGB LED was connected to VCC. See *Figure 2* for full circuit configuration.

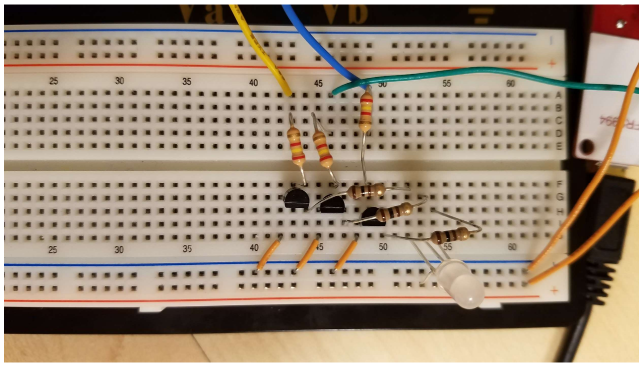


Figure 2: Circuit on Breadboard

4. MSP430FR5994

The MSP430FR5994 was used due to having many features that pertained to the program requirements. The main feature that was needed for this code was a timer with four CCRn registers. The PWM code requires four CCRn registers, and the 5994 has seven CCRn registers on each of its two timers. In addition to the CCRn registers, the 5994 launchpad board comes with a built in supercapacitor. While it isn't used in this code specifically, it wouldn't be very difficult to add supercap use into the code. This would allow the board to run for an extended period of time while unplugged.

5. RGB Node

It was necessary for each node to be able to take the string of HEX values from the master node or a previous node in through UART RX. The nodes were made to be modular, meaning that it can be used alone, but can be easily daisy chained together. Either way the node will perform properly. The first byte contains the size of the package (N). The second, third, and fourth byte were used to set the duty cycle of each LED. The second byte corresponds to the Red LED, the third corresponds to the Blue LED, and the fourth corresponds to the Green LED. The rest of the string had to be repackaged and sent to the next node.*(Appendix 8.1: Table of Description of Bytes 1-N)* *Figure 3* shown below shows how one node should operate.

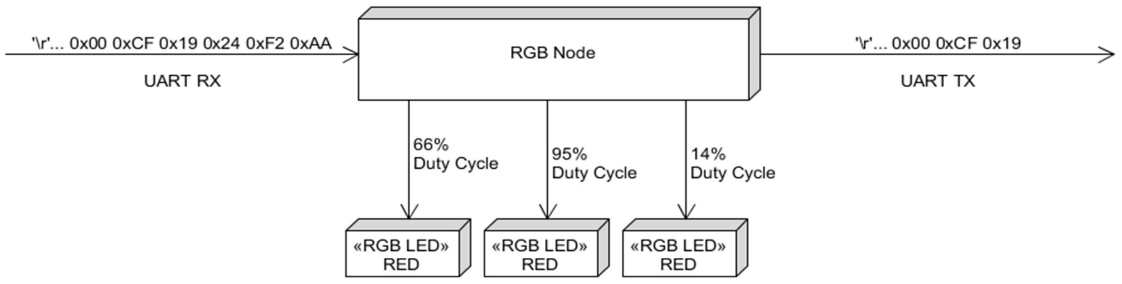


Figure 3: Single Node

The next node will the receive the package sent from the previous node. The first byte will be three less than the previous package, due to the three bytes being taken to set duty cycle of the previous node. The process will continue until until there is no more bytes to pull from. The only bytes that will remain are byte 1 and the last 2 bytes. Byte (N -1) is a carriage return. The complete network of devices should look like the diagram below.

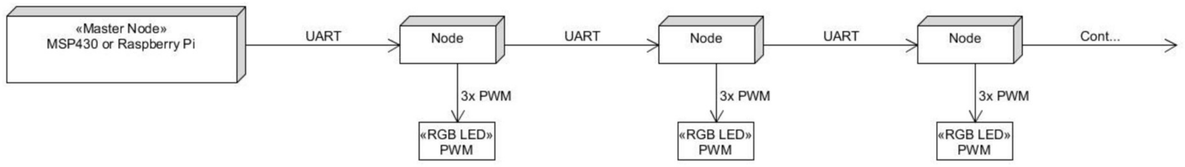
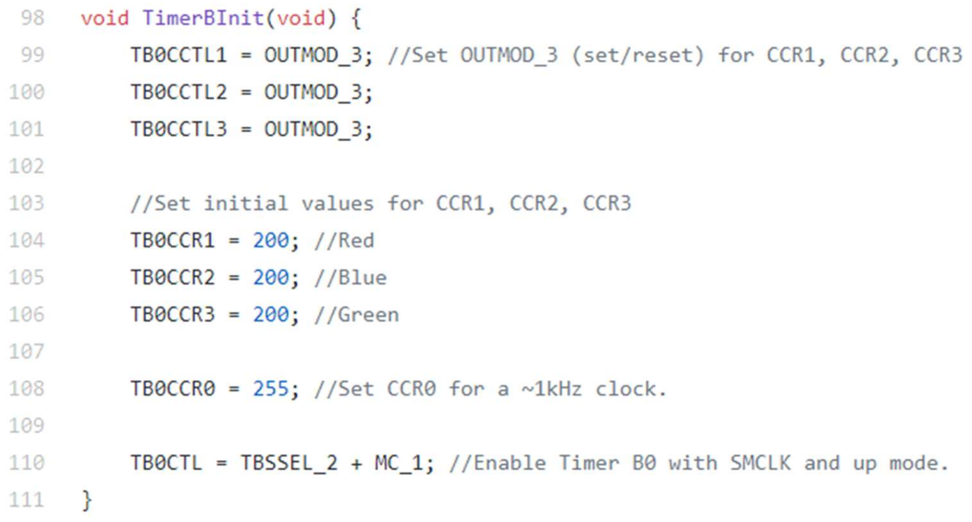


Figure 4: Network of Devices

Program Capabilities

6. PWM

To vary the color of the RGB LED, it is important to be able to send a PWM signal to the LED. In order to generate this signal, the built in clock output mode of the MSP430FR5994 was used. These output modes will trigger an action upon hitting their assigned CCRn register and then trigger a second action upon hitting CCR0. In this instance, three output pins and three CCRn registers were used for each of the colors of the RGB LED. Set/reset mode was used (line 99- 101), so that when the CCRn is hit, the output pin is set high. When CCR0 is hit, the output pin is reset to low. The HEX color values were used as the CCRn value for each of the three RGB output pins, and 255 was used for CCR0 (line 108). Because set/reset mode is used, the duty cycle is 0% when CCRn is equal to CCR0 and 100% when CCRn is zero.



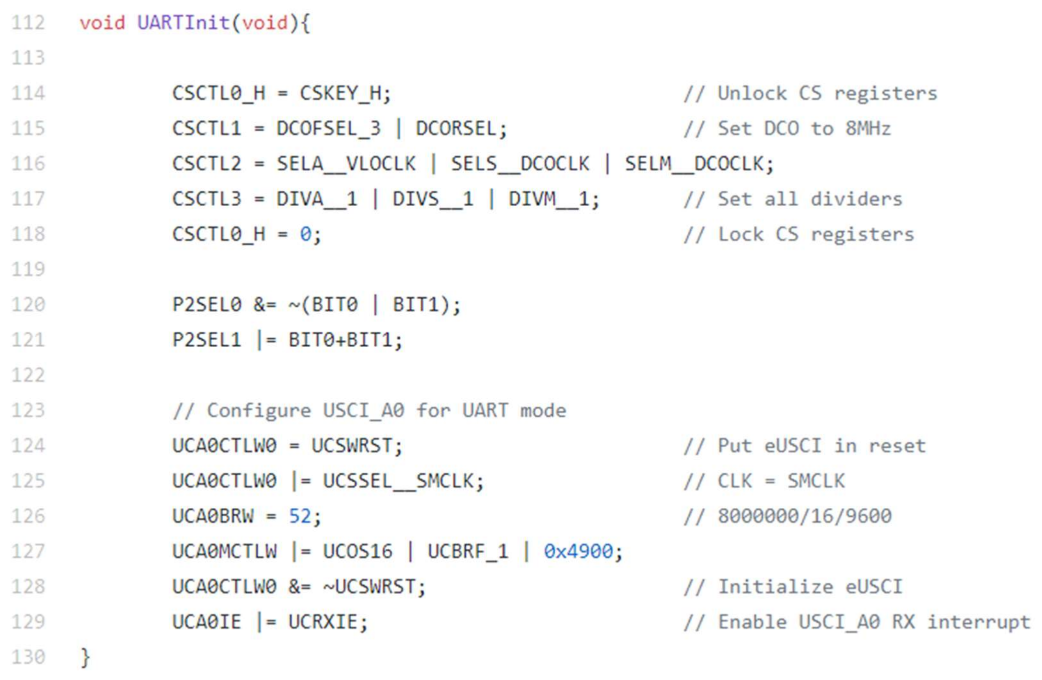
```
 98  void TimerBInit(void) {
 99      TB0CCTL1 = OUTMOD_3; //Set OUTMOD_3 (set/reset) for CCR1, CCR2, CCR3
100      TB0CCTL2 = OUTMOD_3;
101      TB0CCTL3 = OUTMOD_3;
102
103      //Set initial values for CCR1, CCR2, CCR3
104      TB0CCR1 = 200; //Red
105      TB0CCR2 = 200; //Blue
106      TB0CCR3 = 200; //Green
107
108      TB0CCR0 = 255; //Set CCR0 for a ~1kHz clock.
109
110      TB0CTL = TBSSEL_2 + MC_1; //Enable Timer B0 with SMCLK and up mode.
111  }
```

Ports & Capture/Compare Registers used for PWM

RED LED	**P1.4**	**CCR1**
BLUE LED	**P1.5**	**CCR2**
GREEN LED	**P3.4**	**CCR3**

7. UART

The UART code used was based off of the TI- Example code written in 2015 by William Goh. This code showed how to echo incoming UART code. The initialization of UART, using 9600 baud rate, was kept the same for our code (*Figure ##)*, and only had minor adjustments to the UART interrupt vector. Within the software, whenever the receiving flag went up, a different action was chosen depending on the how many items had been received. Case USCI_UART_UCRXIFG within the interrupt was changed to switch between 4 different cases and a default. In case 0, the first byte received was stored as the size of the entire set (line 45). In cases 1-3, the next three bytes were converted into duty cycles (line 49,53, and 57) and put to the LEDs. In case 3, the TX register is also set to (size - 0x03), allowing all bytes except the three used for duty cycle to be repackaged and forwarded to the transfer register. In each case count is incremented to continue to cycle through the switch statement. In the default case TX is set equal to RX. Once the ending byte was transferred, the counter was reset, allowing it to accept the next set. So if count is greater than (size -1) in this case, count is reset to 0.

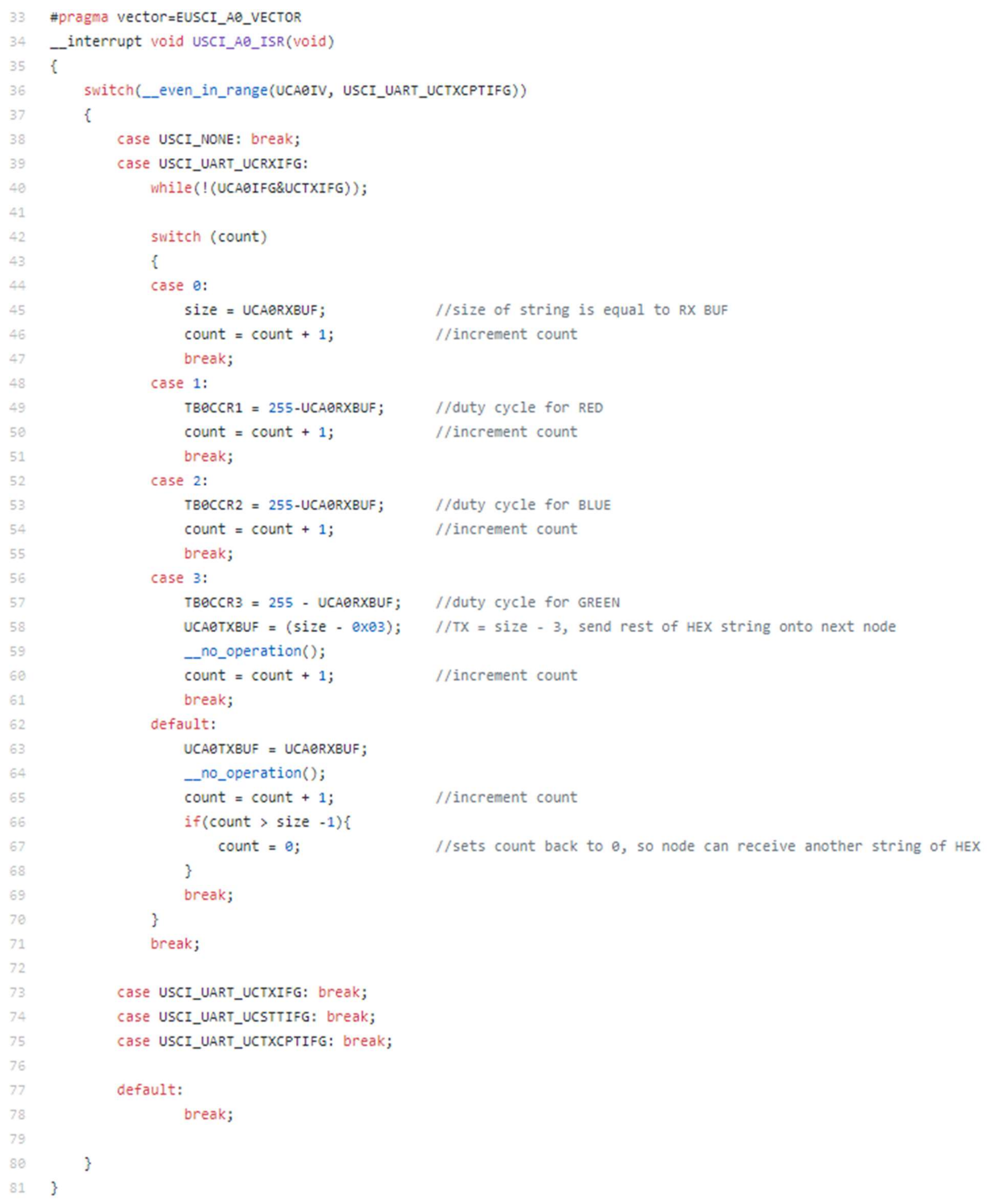


```
void UARTInit(void){

        CSCTL0_H = CSKEY_H;                          // Unlock CS registers
        CSCTL1 = DCOFSEL_3 | DCORSEL;                // Set DCO to 8MHz
        CSCTL2 = SELA__VLOCLK | SELS__DCOCLK | SELM__DCOCLK;
        CSCTL3 = DIVA__1 | DIVS__1 | DIVM__1;        // Set all dividers
        CSCTL0_H = 0;                                // Lock CS registers

        P2SEL0 &= ~(BIT0 | BIT1);
        P2SEL1 |= BIT0+BIT1;

        // Configure USCI_A0 for UART mode
        UCA0CTLW0 = UCSWRST;                         // Put eUSCI in reset
        UCA0CTLW0 |= UCSSEL__SMCLK;                  // CLK = SMCLK
        UCA0BRW = 52;                                // 8000000/16/9600
        UCA0MCTLW |= UCOS16 | UCBRF_1 | 0x4900;
        UCA0CTLW0 &= ~UCSWRST;                       // Initialize eUSCI
        UCA0IE |= UCRXIE;                            // Enable USCI_A0 RX interrupt
}
```



```c
#pragma vector=EUSCI_A0_VECTOR
__interrupt void USCI_A0_ISR(void)
{
    switch(__even_in_range(UCA0IV, USCI_UART_UCTXCPTIFG))
    {
        case USCI_NONE: break;
        case USCI_UART_UCRXIFG:
            while(!(UCA0IFG&UCTXIFG));

            switch (count)
            {
            case 0:
                size = UCA0RXBUF;             //size of string is equal to RX BUF
                count = count + 1;            //increment count
                break;
            case 1:
                TB0CCR1 = 255-UCA0RXBUF;      //duty cycle for RED
                count = count + 1;            //increment count
                break;
            case 2:
                TB0CCR2 = 255-UCA0RXBUF;      //duty cycle for BLUE
                count = count + 1;            //increment count
                break;
            case 3:
                TB0CCR3 = 255 - UCA0RXBUF;    //duty cycle for GREEN
                UCA0TXBUF = (size - 0x03);    //TX = size - 3, send rest of HEX string onto next node
                __no_operation();
                count = count + 1;            //increment count
                break;
            default:
                UCA0TXBUF = UCA0RXBUF;
                __no_operation();
                count = count + 1;            //increment count
                if(count > size -1){
                    count = 0;                //sets count back to 0, so node can receive another string of HEX
                }
                break;
            }
            break;

        case USCI_UART_UCTXIFG: break;
        case USCI_UART_UCSTTIFG: break;
        case USCI_UART_UCTXCPTIFG: break;

        default:
                break;

    }
}
```

8. **Appendix**
   8.1: Description of Bytes

Byte Number	Contents	Example
Byte 0	**Number of bytes (N) including this byte**	**0x50 (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)**

8.2: Full Code

```
//Milestone_1- Stranger Things
//Jessica Wozniak, Ryan Hare, and Timothy Gordon

//Input is RBG
#include <msp430.h>

#define RED BIT4 //Pin 1.4 is the TB0CCR1 output pin.
#define BLUE BIT5 //Pin 1.5 is the TB0CCR2 output pin.
#define GREEN BIT4 //Pin 3.4 is the TB0CCR3 output pin.

int size;
unsigned int count = 0;

void LEDInit(void);
void TimerBInit(void);
void UARTInit(void);

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD; // Stop Watchdog timer
    PM5CTL0 &= ~LOCKLPM5; //Disable HIGH Z mode

    LEDInit(); //Initialize LED pins.
    TimerBInit(); //Initialize Timer B.
    UARTInit(); //Initialize UART

    __enable_interrupt(); //Enable interrupts.
    __bis_SR_register(LPM0 + GIE); // Enter LPM0, interrupts enabled
    __no_operation(); // For debugger
}

#pragma vector=EUSCI_A0_VECTOR
__interrupt void USCI_A0_ISR(void)
{
    switch(__even_in_range(UCA0IV, USCI_UART_UCTXCPTIFG))
    {
        case USCI_NONE: break;
        case USCI_UART_UCRXIFG:
            while(!(UCA0IFG&UCTXIFG));

            switch (count)
            {
            case 0:
                size = UCA0RXBUF;            //size of string is equal to RX BUF
                count = count + 1;           //increment count
                break;
            case 1:
                TB0CCR1 = 255-UCA0RXBUF;     //duty cycle for RED
                count = count + 1;           //increment count
                break;
            case 2:
                TB0CCR2 = 255-UCA0RXBUF;     //duty cycle for BLUE
                count = count + 1;           //increment count
                break;
            case 3:
                TB0CCR3 = 255 - UCA0RXBUF;   //duty cycle for GREEN
                UCA0TXBUF = (size - 0x03);   //TX = size - 3, send rest of HEX string onto
next node
                __no_operation();
                count = count + 1;           //increment count
                break;
            default:
                UCA0TXBUF = UCA0RXBUF;
                __no_operation();
                count = count + 1;           //increment count
                if(count > size -1){
                    count = 0;               //sets count back to 0, so node can receive another
string of HEX
```

```
            }
            break;
        }
        break;
    case USCI_UART_UCTXIFG: break;
    case USCI_UART_UCSTTIFG: break;
    case USCI_UART_UCTXCPTIFG: break;
    default:
            break;
  }
}

void LEDInit(void) {
  //For pin 1.4, 1.5, and 3.4, P1DIR = 1, P1SEL0 = 1, P1SEL1 = 0.
  P1DIR |= RED; //Pin 1.4
  P1SEL1 &= ~RED;
  P1SEL0 |= RED;

  P1DIR |= BLUE; //Pin 1.5
  P1SEL1 &= ~BLUE;
  P1SEL0 |= BLUE;

  P3DIR |= GREEN; //Pin 3.4
  P3SEL1 &= ~GREEN;
  P3SEL0 |= GREEN;
}

void TimerBInit(void) {
  TB0CCTL1 = OUTMOD_3; //Set OUTMOD_3 (set/reset) for CCR1, CCR2, CCR3
  TB0CCTL2 = OUTMOD_3;
  TB0CCTL3 = OUTMOD_3;

  //Set initial values for CCR1, CCR2, CCR3
  TB0CCR1 = 200; //Red
  TB0CCR2 = 200; //Blue
  TB0CCR3 = 200; //Green

  TB0CCR0 = 255; //Set CCR0 for a ~1kHz clock.

  TB0CTL = TBSSEL_2 + MC_1;        //Enable Timer B0 with SMCLK and up mode.
}
void UARTInit(void){

    CSCTL0_H = CSKEY_H;                         // Unlock CS registers
```

```
        CSCTL1 = DCOFSEL_3 | DCORSEL;              // Set DCO to 8MHz
        CSCTL2 = SELA__VLOCLK | SELS__DCOCLK | SELM__DCOCLK;
        CSCTL3 = DIVA__1 | DIVS__1 | DIVM__1;       // Set all dividers
        CSCTL0_H = 0;                               // Lock CS registers

        P2SEL0 &= ~(BIT0 | BIT1);
        P2SEL1 |= BIT0+BIT1;

        // Configure USCI_A0 for UART mode
        UCA0CTLW0 = UCSWRST;                       // Put eUSCI in reset
        UCA0CTLW0 |= UCSSEL__SMCLK;                // CLK = SMCLK
        UCA0BRW = 52;                              // 8000000/16/9600
        UCA0MCTLW |= UCOS16 | UCBRF_1 | 0x4900;
        UCA0CTLW0 &= ~UCSWRST;                     // Initialize eUSCI
        UCA0IE |= UCRXIE;                          // Enable USCI_A0 RX interrupt
    }
```

9. References

UART example code
https://github.com/RU09342/lab-1-intro-to-git-c-and-msp430-jessicanicole/blob/master/Example%20Code/MSP430FR5994/msp430fr599x_euscia0_uart_01.c

Figures of Nodes, network of devices, and Description of Bytes table
https://github.com/RU09342/milestone-1-communicating-with-will-byers-ryan-and-jessica