Milestone 1: Stranger Things Lights

Drexel, Ford, Jukus
Rowan University

October 19, 2017

1 Abstract

In the famous Netflix series ”Stranger Things” Will Byers attempts to communicate with Joyce while in seperate dimensions. Joyce strung lights across the wall with letters written under each light and Will Byers would turn on the lights to communicate between dimensions. This scene will be recreated with the use of the MSP430 and daisy-chained together with other boards through UART to control RGB LEDs with pulse width modulation. Through this recreation, Will Byers will be able to communicate with another dimmension and be able to change the colors of the LEDs anywhere in his message.

2 Background

Ebedded systems would become rather limiting in possibilities without communication and the ability to control analog circuits with digital microprocess outputs.

2.1 PWM

Through the use of pulse width modulators (PWM) the intensity of different colors of an RGB LED can be manipulated. A PWM works in two parts, firstly by using a counter that increments until it hits a maximum and then resets itself to 0. In this case the maximum and counter are the Capture Compare Registers. The Capture Compare Registers are capable of doing just as their name states, holding data that can then be evaluated. This makes it invaluable to the PWM since it can use the hardware of the MSP430 to function. This basic functionality was established in Lab 4 and was modified in order to fit the needs of this PWM.

2.2 UART Transmission

Universal Asynchronous Receiver-Transmitter (UART) is the way the board communicates with other serial devices. Since UART is asynchronous, data transfer is not dependent on clock rate, only baud rate. Baud rate is the rate at which bits are sent between serial devices, for this project, the micro-controller used 9600 baud.

3 Design

By itself an LED is rather meaningless, and RGB LED is even more meaningless without some type of control. This project combines hardware and software to process data, send data through UART and control RGB LEDs.

3.1 Capture Compare Registers

The MSP430F5529 was chosen for this task for a specific reason, it's capture compare registers. Compared to other MSP430 micro controllers, the F5529 has seven capture compare registers (CCRx, where x can be 0 to 6). The first CCR, CCR0, can be used as the maximum while the next three CCR's can be used as the duty cycle modulator for each of the LEDs; where CCR1 is the red LED, CCR2 is the green LED, and CCR3 is the blue LED.

3.2 Data Processing

The Data Processing portion is essential to the project, as it deals with the reception, manipulation, and transmission of the data. All of these tasks take place within an interrupt that triggers when the micro-controller is receiving data. Upon entering the interrupt, the application goes into a state machine. This state machine is based on switch statement controlled by the value of integer NBR. NBR stands for number of bytes received. Tracking the number of bytes received and knowing the format of the incoming data is all that is needed to create the state machine. The incoming data format is listed below.
Somewhere between five and 80 bytes will be received and will need to be processed and output. The 5 bytes that must process are listed below:

- Byte 1: number of bits in package (BIP)
- Byte 2: Red LED pulse width out of FFh
- Byte 3: Blue LED pulse width out of FFh
- Byte 4: Green LED pulse width out of FFh
- Bytes 5 to BIP-1: Data to be transmitted
- Byte BIP: return value

The board will need to delete bytes 2-4, update byte 1 as to the new number of bytes in the sequence and then transmit this appended to the new sequence. For clarification and organization a state machine based on the number of bytes received is used.

- NBR = 1: the first byte is assigned to integer BIP, add (BIP-3) to character array MESSAGE. Increment NBR.
- NBR = 2 to 4: assign byte 2 to CCR1, byte 3 to CCR2, and byte 4 to CCR3. Increment NBR.
- 4 ¡ NBR ¡ BIP: add data to MESSAGE, increment NBR.
- NBR = BIP: add data to MESSAGE, assign MESSAGE to UART output for transmission.

3.3 LED Control

The RGB LED used was a common cathode LED to optimize the ease of use. This circuit is far simpler than the one needed for a common anode, this circuit can be seen in Figure 1. Each LED pin was connected to a resistor connected to power and the cathode was connected to ground. In order to make sure the LEDs had the same intensity the turn-on voltage of each LED was measured. 1.8 V for the red LED, 2.5 V for the green, and 2.6 V for the blue LED. By keeping the current across the LED the same, the intensity will be the same. With a supply voltage of 3.3 V the resistor values were calculated based on these measured drops to be 1.5 $k\Omega$ for the red LED, 800 Ω for the green LED, and 700 Ω for the blue LED. This step is done simply to ensure the correct colors are transmitted, this acts as a white balance for the LEDs.

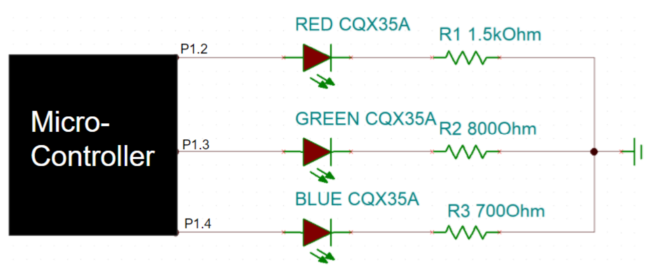

Figure 1: LED circuit broken down to simplest components

As stated previously Bytes 2-4 assign a specific color to one of the LEDs. This is done by assigning the PWM maximum to each CCR. The brightness of each LED is determined by the duty cycle the PWM generates making the LED effectively a hex color wheel.

4 Board Choice

The MSP430F5529 was chosen as the board to implement the design. This decision was made based on the physical layout and pin-out of the micro-controller. The CCR's are laid out on the same port and correspond to their respective bits on the port (P1.1 is CCR0, P1.2 is CCR1 and so on to CCR6). There are additional features that could be later implemented using this board. The multitude of timers, A, B, Master, and Sub-Master clocks, could be used in tandem to perform other tasks or modify which clock is being used. For extra functionality, there are two buttons as well as a temperature sensor, which could be incorporated into this code.

5 Discussion/Conclusions

With the use of the MSP430F5529 Will Byers is not only able to easily communicate with others outside his dimension but he is able to control the color of the lights with which he uses to communicate. This is possible by sending up to 80 bytes through the the UART cable to the first RGB LED that has an MSP430 controller. The controller takes these bytes, interprets them, adjust LEDs as needed, revise the code, and sends the remaining code to the next RGB LED with a controller. This process repeats down the line until all lights have been properly lit conveying the message that was sent.

```
#include <msp430.h>
char message[];
char R = 0;
char G = 0;
char B = 0;
int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                     // Stop WDT
// CCRs stuff
  P1DIR |= BIT2+BIT3+BIT4;                      // P1.2 and P1.3 and P1.4 to gpio/CCRs
  P1SEL |= BIT2+BIT3+BIT4;                      // P1.2 and P1.3 and P1.4 CCR stuff

//Timer Stuff for led
  TA0CCR0 = 512-1;                              // PWM Period is set to 511
  TA0CCTL1 = OUTMOD_7;                          // CCR1 reset/set
  TA0CCR1 = 0;                                  // CCR1 PWM initialization duty cycle
  TA0CCTL2 = OUTMOD_7;                          // CCR2 reset/set
  TA0CCR2 = 0;                                  // CCR2 PWM initialization duty cycle
  TA0CCTL3 = OUTMOD_7;                          // CCR3 reset/set
  TA0CCR3 = 0;                                  // CCR3 PWM initialization duty cycle
  TA0CTL = TASSEL_2 + MC_1 + TACLR;             // SMCLK, up mode, clear TAR

//Timer interrupt
  TA1CCTL0 = CCIE;                             // CCR0 interrupt enabled
  TA1CCR0 = 10000;                              //Aclk runs at 10 hz maybe
  TA1CTL = TASSEL_1 + MC_1;

//UART Jawn

  P3SEL |= BIT3+BIT4;                           // P3.3,4 = USCI_A0 TXD/RXD
  UCA0CTL1 |= UCSWRST;                          // **Put state machine in reset**
  UCA0CTL1 |= UCSSEL_2;                         // SMCLK
  UCA0BR0 = 9;                                  // 1MHz 115200 (see User's Guide)
  UCA0BR1 = 0;                                  // 1MHz 115200
  UCA0MCTL |= UCBRS_1 + UCBRF_0;                // Modulation UCBRSx=1, UCBRFx=0
  UCA0CTL1 &= ~UCSWRST;                         // **Initialize USCI state machine**
  UCA0IE |= UCRXIE;                             // Enable USCI_A0 RX interrupt
                                                // Enable USCI_A0 RX interrupt

  __bis_SR_register(LPM0_bits);                 // Enter LPM0
  __no_operation();                             // For debugger
}
```

Figure 2: Initialization for the each interrupt and ports

```
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)
{
   switch(R)//Checks to see if Red led is set
   {
   case 0 : //if not set then set it to receiving bit
       R = UCA0RXBUF;
   default //leave statement to set the next color
       break;
   }
   switch(G)//checks if Green LED is set
   {
   case 0 : //if not set
       G = UCA0RXBUF;
   default //if it is then leave
       break;
   }
   switch(B)//checks if blue led is set
   {
   case 0 :
       B = UCA0RXBUF;
   default //if not leave
       break;
   }
}

#pragma vector = TIMER1_A0_VECTOR           //Timer counts
__interrupt void TA1_ISR(void)
    {//Fetch LED values from uart
    TA0CCR1 = R;                            // CCR1 PWM duty cycle
    TA0CCR2 = G;                            // CCR2 PWM duty cycle
    TA0CCR3 = B;
    }
```

Figure 3: Interrupt Service Routines for UART and Timer A