Stranger Things Light Wall

Shani Thapa and Mitchell Hay
Rowan University

October 17, 2017

1 Abstract

The purpose of this project was to build an ”Addressable” RGB LED. Students were to able choose any of the MSP430 development boards they owned to build this milestone, provided they explained their reasoning. The ”Addressable” RGB LED must be able to receive a long string, read the amount of bytes in the message, remove three values to light up an RGB LED, and finally transmit the rest of the message onto the next board. For this purpose, UART was used for the communication.

2 Introduction

The broad scope of this milestone is that a master node will first transmit a message through UART onto the MSP430 boards used by the students. The message will be a long string of 80 bytes containing hex values. The students then have to receive that message, count the number of bytes, take three values to convert into a pulse width modulation output, and then transmit the rest of the message to the next person. The code should be robust enough that it will work regardless of if it receives the message first, last, or in the middle. Communication between boards is the major issue in this exercise. We must be able to configure our board to receive and transmit messages.

3 Background

Unless the students were doing UART extra work for the previous labs, there are new variables and registers students have to manipulate for UART communication. The initial setup revolved mainly around UART and the code used for the Hardware PWM. Since the Hardware PWM has already been completed, that will not be discussed in the background. First, the UCA0TxD and UCA0RxD must be selected through the PxSEL bits, a process similar when setting the timer in Hardware PWM. These are the

receive and transmit buffers for UART communication. Each board will have different pins for each of their receive and transmit buffers. There may even be multiple buffers on one board. The board we used was the F5529, which has multiple buffers. We decided to use USCA0, which correspond to Pins 3.3 and 3.4. Figure 1 shows the pin functions for all of Port 3 for F5529 and how to select each peripheral through the PxDIR and PxSEL bits. The datasheet for the board used must be checked to find which pins the transmit and receive buffers are located in.

Table 6-48. Port P3 (P3.0 to P3.7) Pin Functions

PIN NAME (P3.x)	x	FUNCTION	CONTROL BITS OR SIGNALS[1]	
			P3DIR.x	P3SEL.x
P3.0/UCB0SIMO/UCB0SDA	0	P3.0 (I/O)	I: 0; O: 1	0
		UCB0SIMO/UCB0SDA[2] [3]	X	1
P3.1/UCB0SOMI/UCB0SCL	1	P3.1 (I/O)	I: 0; O: 1	0
		UCB0SOMI/UCB0SCL[2] [3]	X	1
P3.2/UCB0CLK/UCA0STE	2	P3.2 (I/O)	I: 0; O: 1	0
		UCB0CLK/UCA0STE[2] [4]	X	1
P3.3/UCA0TXD/UCA0SIMO	3	P3.3 (I/O)	I: 0; O: 1	0
		UCA0TXD/UCA0SIMO[2]	X	1
P3.4/UCA0RXD/UCA0SOMI	4	P3.4 (I/O)	I: 0; O: 1	0
		UCA0RXD/UCA0SOMI[2]	X	1
P3.5/TB0.5[5]	5	P3.5 (I/O)	I: 0; O: 1	0
		TB0.CCI5A	0	1
		TB0.5	1	1
P3.6/TB0.6[5]	6	P3.6 (I/O)	I: 0; O: 1	0
		TB0.CCI6A	0	1
		TB0.6	1	1
P3.7/TB0OUTH/SVMOUT[5]	7	P3.7 (I/O)	I: 0; O: 1	0
		TB0OUTH	0	1
		SVMOUT	1	1

Figure 1: Pin Functions for P3

The UCAxCTLx register then has to be manipulated, which decides which clock the UART will use. UCSSELx selects the clock itself. UCAxBR0 and UCAxBRO1 allows the clock to be divided to the desired value. UCAxMCTLx modulates the frequency if required. UCBRS and UCBRF determine the modulation. UCSWRST controls the USCI. UCRXIE must be enabled for the interrupt to be generated once a message is received. The following will list all the variables that were manipulated in our code along with its function:

- UCAxCTLx has two registers. One for protocol which isn't used. The other for configuring clocking, enable, interrupts, etc. The second one was used extensively.
  - Ex. $UCA0CTL1| = UCSSEL\_1$ // clock is set to ACLK
- UCSWRST is the Module software reset. USCI is in reset by default and has to be cleared to use the module.
- UCSSELx is the clock source select. The x can be 0-3 to set different clocks.
  - Ex. $UCSSEL_2$ // SMCLK
- UCAxBROx are two registers that control the baud rate and could be used for modulation

- UCAxBRSx is the first modulation stage select
- UCAxBRFx is the second modulated stage select
- UCRXIE is the interrupt enable for when UART receives a message

After those variables were configured, the UART setup was finished and the ISR needed to be populated. The ISR does the bulk of the work in this milestone, and will be explained in a later section.

3.1 MSP430F5529

We originally thought that we needed three timers to accomplish this milestone; one for each output to the PWM of the RGB LED. We later learned we only needed multiple CCR instead of timers, but, under the original assumption we automatically cut the G2553 and FR2311 which only had 2 timers each. Then, we were left with the FR6989, F5529, and FR5994. Eventually, we decided to use the F5529 for three reasons. First, the board had all the timers and CCR registers reqired to complete the milestone. Second, compared to the FR6989 and FR5994, it's cheaper in price. Finally, Mitchall had been doing UART extra work from the labs, and he had success using the F5529. So we decided to roll with the F5529.

4 Evaluation and Results

4.1 PWM

Setting up the pins and timers was the first priority. Timer A0 was set up such that the sub-main clock was used in up mode. Up mode means that the timer would count up to the value in the first capture compare register, CCR0, then reset to zero. Setting up the timer in this way was essential for the functionality of the pulse width modulation. The MSP430F5529 has 5 capture compare registers for Timer A0, so four of them would be used for this project. The first capture compare register, CCR0, would be used as a baseline for the rest of the capture compare registers. The highest value the timer could count up to is the value in CCR0, so the other three registers were compared to this value. CCR1, CCR2, and CCR3 were each used for one color in the RGB LED. Since the maximum value that could be received is 255, CCR0 was set to 255. This made it easy to assign values from the input string to the other capture compare registers.

The capture compare registers each corresponded with GPIO pins on the MSP430F5529. Pins 1.2, 1.3, and 1.4 are used for Timer A0 CCR1, CCR2, and CCR3, respectively. This can be seen in the MSP430F5529 pinout diagram. These three pins are set as outputs, and the peripheral function is selected. We have included Figure 2 for the Pin Functions of Port 1 on the F5529. The pin select is done by the following lines of code:

```
// PWM Setup
P1DIR |= BIT2 + BIT3 + BIT4;          // Set pins as outputs
P1SEL |= BIT2 + BIT3 + BIT4;          // Set pins to CCRx values
```

Table 6-46. Port P1 (P1.0 to P1.7) Pin Functions

PIN NAME (P1.x)	x	FUNCTION	CONTROL BITS OR SIGNALS	
			P1DIR.x	P1SEL.x
P1.0/TA0CLK/ACLK	0	P1.0 (I/O)	I: 0; O: 1	0
		TA0CLK	0	1
		ACLK	1	1
P1.1/TA0.0	1	P1.1 (I/O)	I: 0; O: 1	0
		TA0.CCI0A	0	1
		TA0.0	1	1
P1.2/TA0.1	2	P1.2 (I/O)	I: 0; O: 1	0
		TA0.CCI1A	0	1
		TA0.1	1	1
P1.3/TA0.2	3	P1.3 (I/O)	I: 0; O: 1	0
		TA0.CCI2A	0	1
		TA0.2	1	1
P1.4/TA0.3	4	P1.4 (I/O)	I: 0; O: 1	0
		TA0.CCI3A	0	1
		TA0.3	1	1
P1.5/TA0.4	5	P1.5 (I/O)	I: 0; O: 1	0
		TA0.CCI4A	0	1
		TA0.4	1	1
P1.6/TA1CLK/CBOUT	6	P1.6 (I/O)	I: 0; O: 1	0
		TA1CLK	0	1
		CBOUT comparator B	1	1
P1.7/TA1.0	7	P1.7 (I/O)	I: 0; O: 1	0
		TA1.CCI0A	0	1
		TA1.0	1	1

Figure 2: Pin Functions for P1

After that, the CCRx of the timer were set to the integers r, g, b for the colors. Then the timer CCTLx had to be set to OUTMOD_7 which does two things. First, the output is reset when it counts to the TxCCRx value. Second, the output is set when it counts to the TxCCR0 value. This switching between resetting and setting will create a PWM signal. This signal will be modulated by the timers to create the intended duty cycle.

4.2 UART

The UART function was then written to make sure that the board could be communicated with. Texas Instruments gave a code example on UART, however there were a few changes that needed to be made in order for the program to work for this project. The alternate clock was used and the baud rate divider was changed to ensure that data was being sent and received at 9600 bps. A simple loopback program was tested, and using Realterm, the board was able to send and receive characters at 9600 bps.

The project stated that the board was to be run in low power mode until the string of bytes was sent to it. This means that all of the functionality beyond the initial setup had to be done in the UART interrupt service routine. Before writing the ISR, there were a few variables that needed to be declared: byte counter, total number of bytes, number for red LED, number for green LED, number for blue LED, integer array for message, and two counters.

The byte counter kept track of what number byte was being received at the time. The total number of bytes took the first received byte, which was used to tell how

many total bytes are in the string. The three following bytes would be used for the red, green, and blue LEDs, respectively. The numbers for the red, green, and blue LED would determine each of their duty cycles, and thus control the color. The integer array was used to store the outgoing string to the next board. The first number of the outgoing message was the first number of the incoming message minus three. This is because there would be three bytes omitted from the outgoing message, which would be the next three bytes. Every byte after that would be added to the message, then transmitted to the next board.

4.3 RGB Circuit

The RGB LED that was used in the lab was of the common anode type. This type of LED must have the anode connected to power, and when the pins of the Red, Green, and Blue LEDs have a path to ground, it will turn on. As a result, we needed to make a circuit with three MOSFETs that would allow current to flow to ground only when the PWM values from the F5529 were received. The drain pins of each of the MOSFETs were connected to the pins of the RGB LED. The gate pins were connected to the PWM outputs from the F5529, and the source pins were connected to ground. This would allow the circuit to act in the way stated in the milestone requirements. For example, when the PWM output of pin 1.2 was on, then it would turn the MOSFET on and allow current to flow through the red LED. The Figure 3 shows the circuit diagram used to turn the LEDs on.

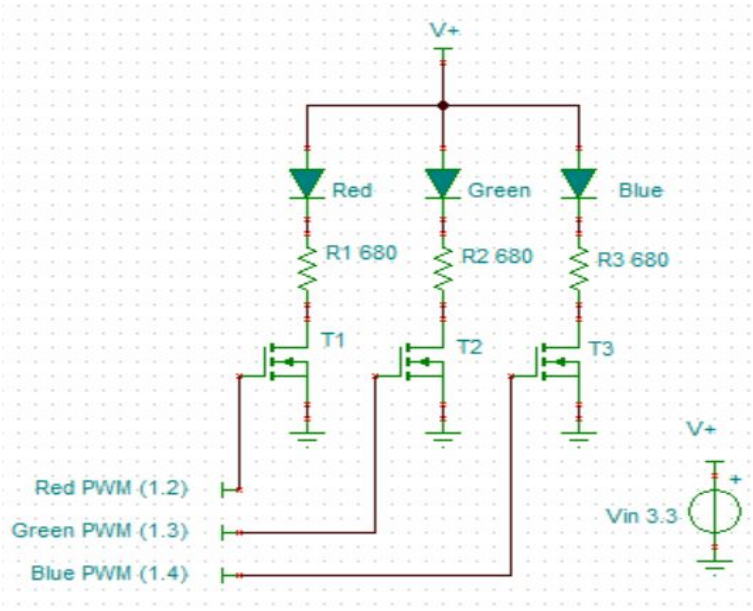

Figure 3: SPICE Drawing of RGB Circuit

5 Discussion/Conclusions

The milestone project worked exactly as expected, displaying a color on the RGB LED and transmitting the rest of the string. There were a wide range of color combinations that worked well with the circuit setup. All of the code can be seen in Section 6.1 in the Appendix.

The string input and output was the most important part of the lab. This was handled in the UART interrupt service routine. The RGB circuit was developed around this ISR, such that the the string input could be directly assigned to the capture compare registers, which would change the duty cycle of each LED. The rest of the string was also able to be transmitted correctly, as confirmed by Realterm. This will be important when multiple boards are connected together. The rest of the code was very simple, as it was just initializing timers and outputs.

As shown in Figure 3, this circuit was made using three 680Ω resistors and three 2N700 MOSFETS. These transistors allowed for the circuit to work how the task desired. In order to decrease costs, the lab team discussed connecting the resistors from the red, green, and blue pins directly to the outputs of the PWM pins. This worked, but not very well. When the LED was supposed to be off, there was still some color being displayed. This was because there was current leaking through the LED, which caused it to glow dimly. Using the MOSFETs really cleaned up the color and turned the LED off when it should have been off.

One strange error that we never solved was the SMCLK, sub main clock of the board. We initially wanted to use the 1MHz clock and divide it to get our 9600 Baud rate. However, it was not responding to inputs from PUTTY. We switched over to the ACLK which originally runs at around 32kHz. After dividing that clock and running the code, it was working correctly so we decided to just use the ACLK. We are not sure why SMCLK did not work. Overall, the milestone was able to be completed successfully.

6 Appendix

6.1 Milestone 1 Code

```
1  /*
2   * Mitchell Hay and Shani Thapa
3   * Milestone 1 - Stranger Things Light Wall
4   * RU09342
5   * MSP430F5529
6   */
7
8  #include <msp430.h>
9
```

```
10  volatile unsigned int i = 0;
11  volatile unsigned int j = 0;
12  volatile unsigned int r = 0;
13  volatile unsigned int g = 0;
14  volatile unsigned int b = 0;
15  volatile unsigned int byteCnt = 0;
16  volatile unsigned int numBytes = 0;
17  int message[80];
18  int rx;
19
20  void main(void) {
21          // Clock Setup
22          WDTCTL = WDTPW + WDTHOLD;           // Stop WDT
23          TA0CCR0 = 256 - 1;                  // PWM Period
24          TA0CTL = TASSEL_2 + MC_1 + TACLR;   // SMCLK, up mode, clear TAR
25
26          // PWM Setup
27          P1DIR |= BIT2 + BIT3 + BIT4;        // Set up Pins 1.2,3, and 4 as outputs
28          P1SEL |= BIT2 + BIT3 + BIT4;        // Set Pins 1.2,3, and 4 to Timer A0 CCRx
29          TA0CCTL1 = OUTMOD_7;                // CCR1 reset/set
30          TA0CCR1 = r;                        // CCR1 PWM duty cycle, red LED
31          TA0CCTL2 = OUTMOD_7;                // CCR2 reset/set,
32          TA0CCR2 = g;                        // CCR2 PWM duty cycle, green LED
33          TA0CCTL3 = OUTMOD_7;                // CCR3 reset/set
34          TA0CCR3 = b;                        // CCR3 PWM duty cycle, blue LED
35
36          // UART Setup
37          P3SEL |= BIT3 + BIT4;               // P3.3,4 = USCI_A0 TXD/RXD
38          UCA0CTL1 |= UCSWRST;                // **Put state machine in reset**
39          UCA0CTL1 |= UCSSEL_1;               // ACLK
40          UCA0BR0 = 3;                        // 32726 MHz/3 = 9600 (see User's Guide)
41          UCA0BR1 = 0;                        // 1MHz 3
42          UCA0MCTL |= UCBRS_3 + UCBRF_0;      // Modulation UCBRSx=1, UCBRFx=0
43          UCA0CTL1 &= ~UCSWRST;               // **Initialize USCI state machine**
44          UCA0IE |= UCRXIE;                   // Enable USCI_A0 RX interrupt
45
46          __enable_interrupt();                           // Enable Interrupts
47          __bis_SR_register(LPM0 + GIE);      // Enter LPM0, interrupts enabled
48  }
49
50  // Transmit and Receive Interrupts
51  #if defined(__TI_COMPILER_VERSION__) || defined(__IAR_SYSTEMS_ICC__)
52  #pragma vector=USCI_A0_VECTOR
53  __interrupt void USCI_A0_ISR(void)
54  #elif defined(__GNUC__)
55  void __attribute__ ((interrupt(USCI_A0_VECTOR))) USCI_A0_ISR (void)
```

```
#else
#error Compiler not supported!
#endif
{
switch (__even_in_range(UCA0IV, 4)) {
        case 0:
                break;                          // Vector 0 - no interrupt
        case 2:                                 // Vector 2 - RXIFG
                while (!(UCA0IFG & UCTXIFG));
                rx = UCA0RXBUF;                 // Hold received character
                // Check first number of message
                if (byteCnt == 0) {
                        numBytes = rx;
                        byteCnt++;
                        message[i] = numBytes - 3;
                        i++;
                }

                // Assign RGB values to CCRs
                else if (byteCnt >= 1 && byteCnt <= 3) {
                        switch (byteCnt) {
                        case 1:
                                r = rx;           // Set red to the 2nd int received
                                TA0CCR1 = r;      // CCR1 PWM duty cycle, red
                                byteCnt++;        // Increment byte count
                                break;
                        case 2:
                                g = rx;           // Set green to 3rd int received
                                TA0CCR2 = g;      // CCR2 PWM duty cycle, green
                                byteCnt++;        // Increment byte count
                                break;
                        case 3:
                                b = rx;           // Set blue to 4th int received
                                TA0CCR3 = b;      // CCR3 PWM duty cycle, blue
                                byteCnt++;        // Increment byte count
                                break;
                        }
                }

                // Write all of the other values
                else if (byteCnt < numBytes) {
                        // Add to message to send out
                        message[i] = rx;
                        i++;
                        byteCnt++;
                        // If the message is over, transmit to next person
```

```
102                             if (byteCnt == numBytes) {
103                                     while (message[j] != 0x0D) {
104                                             while (!(UCA0IFG & UCTXIFG))
105                                                     ;
106                                             UCA0TXBUF = message[j];
107                                             j++;
108                                     }
109                                     while (!(UCA0IFG & UCTXIFG))
110                                             ;
111                                     UCA0TXBUF = 0x0D; // Add to end of message
112                             }
113                     }
114                     break;
115             case 4:
116                     break;
117             default:
118                     break;
119             }
120     }
```