Milestone #1: Stranger Things RGB LED

Tyler Brady and Mike Giuliano
Rowan University

October 17, 2017

1 Abstract

Inter-dimensional communication is no easy feat. In Netflix's "Stranger Things", this was accomplished using Christmas lights strung up to correspond to letters of the alphabet. An incoming message would blink the letters in the message one at a time in the order of how they appeared. Inspired by such fiction, this application note explores the implementation of an addressable LED node that blinks according to an input pattern of hex values. It is achieved using the MSPF5529 micro-controller and supports connection in series with other nodes to comprise the full alphabet. Unfortunately, based on design restrictions, communication is limited to this dimension.

2 Introduction/Background

The purpose of this lab was to create a program that could be loaded onto an MSP430 embedded processor to control an RGB LED based on input. In order to be capable of handling this lab, students must understand UART, TIMERS, PWM, and GPIO. In order to control the LED a starting message will be sent that consists of 79 bytes. The first byte is the length byte, this byte will contain the overall length of the message and will decrease by a rate of 3 bytes every board. The 77 bytes after that contain the RGB information for each stage of the chain in the order of: Red, Green, and Blue. This pattern repeats until it reaches the last stage which contains the byte $0x0D$ which is the return byte. In order to successfully implement the program, the board must be programed in a manner than ensures it will work no matter its position within the chain. By the end of the chain the return should be the remaining length and return byte.

3 Implementation

3.1 Processor Selection

With a total of five boards, selecting the right processor is a difficult task. For the program we chose to use the $MSP430F5529$. The main reasoning behind this board choice was the amount of capture/control registers available for use. In this processor there are a total of five CCRs available for use, making the implementation of the PWM much easier as only one timer needs to be made. The $MSP430G2553$ was not chosen as code composer seems to have an issue with some boards and recognition. The $MSP430FR2311$ was written off for its FRAM size. The $MSP430FR6989$ was not used due to its higher cost, and lack of need for the display peripheral. Finally, the $MSP430FR5994$ was not chosen because of the lack of need for the super capacitor. Overall the 5529 fit the current needs of the project with out adding too many unnecessary peripherals.

3.2 PWM

Pulse width modulation is accomplished by controlling the separate duty cycles for the red, blue, and green diodes to produce an effective color and brightness. Implementation-wise, this process takes advantage of several features of the MSPF5529's architecture, namely its four capture compare registers built into Timer A0 that eliminate the need for additional timers. This timer is set to SMCLK, basing its frequency at 1 MHz, and is configured to count in UP mode. The clock divider ID_2 makes the effective clock rate 1 kHz. TA0CCR0 determines the clock period while the three additional capture compare registers associated with Timer A0 are configured in Set/Reset mode using $OUTMOD_7$. This means that when the timer reaches one of the values set in a CCR, the associated pin is set high. The timer continues until it reaches the the clock period value stored in TA0CCR0, at which point an overflow occurs and the pin is reset to its initial state 0 specified by P1OUT. The ratio of this on and off state for each red, blue, and green diode determines the pulse width of the signal.

```
//PWM TIMER SETUP
TA0CCTL1 = OUTMOD_7;//SET/RESET MODE RED
TA0CCTL2 = OUTMOD_7;//SET/RESET MODE GREEN
TA0CCTL3 = OUTMOD_7;//SET/RESET MODE BLUE
TA0CCR0 = 255; // Full Cycle
TA0CCR1 = 0; //Red set at 0%
TA0CCR2 = 0; //Green set at 0%
TA0CCR3 = 0; //Blue set at 0%
TA0CTL = TBSSEL_2 + MC_1 + ID_2; //SMCLK/4, up mode
```

Figure 1: Pulse Width Modulation code

3.3 UART Implementation

In order for data to be passed in and out of the device, the Universal Asynchronous Receiver/Transmitter or UART for short must be set up correctly. This allows the processor to receive data such as hex values to be used within the processor, effectively allowing the user to communicate with a program. In this slice of the program, the first step is to set up the pins in which the UART transmits and receives from. For the 5529 the TX pin peripheral is located on P3.3 and RX is located on R3.4. The next step is to disable the state machine from being activated by putting it into a constant state of reset. Then according to the needed baudrate and modulation, values are set into their corresponding registers. The last step is to re-enable the state machine and enable the RX interrupt. The exact registers and macros for the MSP430F5529 can be found below in Fig. 2. Meanwhile the baudrate settings must be calculated to be 9600bps through the online calculator found in the code as well.

```
//UART SETUP
P3SEL |= BIT3; //TX peripheral mode
P3SEL |= BIT4; //RX peripheral mode
UCA0CTL1 |= (UCSWRST); //State machine reset + small clock initialization
UCA0CTL1 |= UCSSEL_2;
UCA0BR0 = 6; //9600 baud *Determined by TI calculator(http://processors.wiki.ti.com/index.php/USCI_UART_Baud_Rate_Gen_Mode_Selection)
UCA0BR1 = 0; //9600 baud *Determined by TI calculator(http://processors.wiki.ti.com/index.php/USCI_UART_Baud_Rate_Gen_Mode_Selection)
UCA0MCTL |= UCBRS_0; // Modulation
UCA0MCTL |= UCBRF_13;
UCA0MCTL |= UCOS16;
UCA0CTL1 &= ~UCSWRST; // **Initialize USCI state machine**
UCA0IE |= UCRXIE; // Enable USCI_A0 RX interrupt
//
```

Figure 2: UART code

3.4 State Machine

In this implementation, the state machine is used as the logic to control what LED's PWM is being modified, and if anything is being passed along to the next board. The state machine contains five total states that are set up to only trigger when receiving of data. The first thing the processor should receive is the message length byte, therefore the first state is set to handle decrementing the length by three and passing that information along. This operation is quickly followed by a no operation to allow the processor to send the data without missing the next byte. The second state handles adjusting the Red LED PWM timer based on the incoming byte. The third handles the Green LED and the fourth handles the Blue LED. The last state handles the remaining bytes that are for the next processors down the line. Finally, the last step is to wait for the return byte, which resets the state bit to zero to prepare for another message.

```
switch(bitcounter)
{
    case 0:
        while(!(UCA0IFG & UCTXIFG)); //As long as you're not already transmitting, continue, else trap until not transmitting.
        UCA0TXBUF = UCA0RXBUF - 3; //Transmits the length of the remaining bytes
        __no_operation(); //Pauses the clock for a moment
        break;
    case 1:
        TA0CCR1 = (UCA0RXBUF); //SETS RED SECTION
        break;
    case 2:
        TA0CCR3 = (UCA0RXBUF); //SETS GREEN SECTION
        break;
    case 3:
        TA0CCR3 = (UCA0RXBUF); //SETS BLUE SECTION
        break;
    default:
        while(!(UCA0IFG & UCTXIFG)); //Repeat command from case 0.
        UCA0TXBUF = UCA0RXBUF; //Just transmit the incoming byte on to the next board
}
```

Figure 3: State machine code

3.5 Off-board circuit

3.5.1 Common Anode vs Cathode

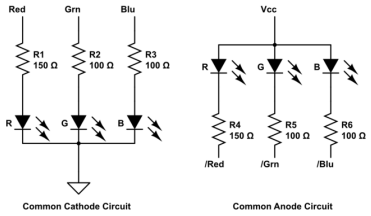

Figure 4: Common Anode vs Common Cathode

In Fig. 4 above, the differences between common anode and cathode can be seen to be the direction of the diodes. In common anode, the input to the LED diodes are all set to a high of $3.3V$, meaning in order to control the state of the three colors a MOSFET is needed on the output of each diode. This allows the PWM outputs to control when each LED is activated.

3.5.2 Circuit Model

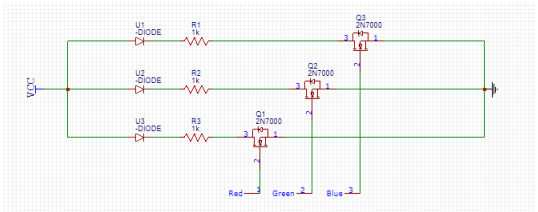

Figure 5: Spice model of off-board circuit

Pictured above in Fig. 5 is the spice circuit model used in the creation of the circuit. Each NMOS is connected to the output of its respective PWM pin, ranging from PIN 1.2, 1.3, and 1.4. These pins corresponded directly to RED, GREEN, and BLUE LED diodes. Since the output of the pins controlled only the gates of the MOSFETS, negligible current should be drawn from the output of each pin and therefore should not be an issue for the board.

3.5.3 Off-Board Connections

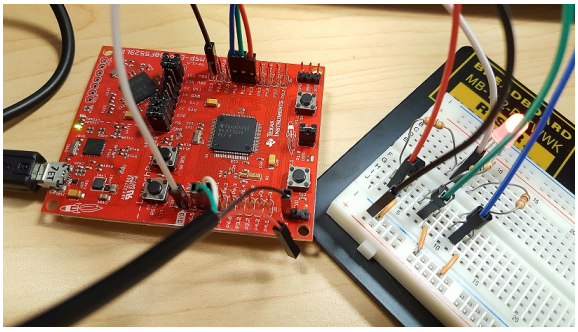

Figure 6: Real circuit implementation

In Fig. 6 above, the connections between breadboard and MSP430 can be seen. The MSP430's power supply was used to power the board, and each of the pins listed above were connected to their corresponding gates. Also pictured is the connection points for the USB to serial cable used to send in the message values to the board.

4 Discussion/Conclusions

The successful implementation of an RGB LED that outputs a specific color based on an input string of hex values requires a working knowledge of several core components. These are: timer configuration, pulse-width modulation, and communication via the UART serial interface. Although applied to the MSP430F5529 architecture, these concepts are adaptable to any family of micro-controllers and for a wide array of specific-use scenarios.

5 References

Much of the work for this project was done in tandem with Kevin Miskovich and Marc Giordano, therefore much of the code is highly similar between the two groups.

MSP430F5529 Datasheet:
http://www.ti.com/lit/ug/slau533d/slau533d.pdf