Milestone 1: Communicating with Will Byers

D. Boorstein and B. Nugent
Rowan University

October 17, 2017

1 Abstract

An RGB LED node which accepts a string of hex values and produces a pulse-width modulated output signal was implemented using the MSP430FR6989 microprocessor. The node could be chained with similar RGB LED nodes to produce complex signals. This was done to emulate the famous scene in the Netflix©series *Stranger Things* in which the character Will Byers communicates to his mother using a string of lights on a wall.

2 Introduction

The Universal Asynchronous Receiver-Transmitter (UART) device embedded in the MSP430 family of devices is useful for human-machine interfacing. UART enables serial communication between an operator and the hardware, allowing the operator to send a string of recognizable characters to the microprocessor and produce a unique output.

Probably the simplest task to perform using UART is character echoing. The hardware UART in the microprocessor can be configured to transmit the characters it receives. A perhaps more challenging task is to transmit only a set of the received characters and perform a separate task in parallel.

In this milestone project, the objective is to emulate a famous scene in the Netflix©series *Stranger Things* using RGB LEDs and RGB LED driver nodes. This is implemented using the MSP430FR6989 microcontroller, or a similar microcontroller, for each node.

The scene is created using a network of devices, each of which creates exactly one node and controls one LED. This network is displayed in Figure 1.

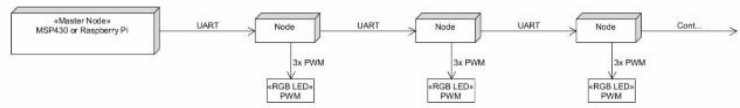


Figure 1: RGB Network

Each node will receive a string of hex values from its preceding node, in the UART RX buffer, and use the four least significant bytes for its own processing. Three of these bytes provide the RGB values for the node while the other byte indicates the total length of the string. The RGB values are used to alter the duty cycle of the signals controlling each color. The node will then transmit the remainder of the hex values to the next node, with the updated string length, through the UART TX buffer.

3 Background

Character echoing through hardware UART was explored in Lab 1: Introduction to C, Git, and the MSP430. The example code provided initialized proper registers to enable UART on a series of MSP430 devices. It would have been possible, using serial communication software, to send a string of characters to the microprocessor's UART module receiver. The USB-to-UART serial converter would receive the signal and echo back the string of characters in the software's terminal window. Unfortunately, the example code was flawed: no characters were returned from the microprocessor. However incorrect the code, it nonetheless served as a good starting point for implementing UART.

The steps for initializing UART are not much different than those for GPIO or timers. Like these, UART has its own interrupt vector registers and associated interrupt flags. Many product lines in the MSP430 family of devices contain a Universal Serial Communication Interface (USCI), a UART module embedded in the hardware. USCI architecture is generally shared among the devices, however some contain an updated module, the Enhanced USCI (eUSCI).

3.1 MSP430FR6989

The MSP430FR6989 provides all of the requirements to implement this application and more. The following criteria were desired:

- an accessible UART module
- a timer module with a sufficient amount of capture/compare registers
- an internal clock module

The FR6989 is one example of an MSP430 that has an eUSCI module. It also has an LCD screen. With additional code, it could be possible to readout the received string

of UART characters.

The UART module must also allow communication over the UART channel to alter the RGB values and transmit the string to the next RGB node. Additionally, it is helpful to have at least four capture/compare registers on a single timer module. Using a single timer module is preferable over having to initialize multiple timer modules and capture/compare registers.

3.2 USCI and eUSCI Modules

The eUSCI module is essentially the updated version of the USCI, and can be found in the MSP430FR2311, FR5994, and FR6989 models. The eUSCI also has some additional features, including:

- enhanced baud rate calculation;
- selectable deglitch filter lengths;
- additional transmit completion and reception start registers.

Additionally, there have been many changes to the word lengths of registers. Some registers have been changed from 8-bit word size to 16-bit word size, and some registers have been combined. Finally, all registers have been added to a single UART interrupt vector register (UCAxIV)[2].

The eUSCI, for this application, is used in UART mode. This allows for communication from the host PC to the microcontroller from a serial communication program, such as RealTerm or PuTTY.
The general UART connection is displayed in Figure 2.

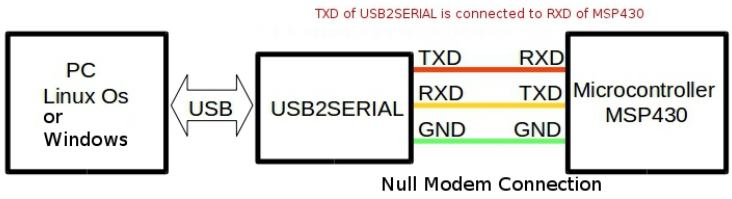


Figure 2: UART Connections

3.3 Pulse-Width Modulation and LED's

The luminosity of an LED can be controlled by altering the duty cycle of the constant current source which drives the LED. By setting a 50% duty cycle, for example, the LED is on for half of the time and off for half of the time. However, with a frequency greater than around 70Hz, the LED's toggle is not perceived by the average human

eye. Instead, the LED is perceived to be 50% as bright as it is at a 100% duty cycle.

Using an RGB LED, the intensity of the red, green and blue contributions to the overall luminosity can be altered to create various colors. The functionality of an RGB LED does not differ much from a conventional LED. It is important to note, though, that the LED used is a common-anode LED. This means that the common pin is connected to the V_{cc} pin, which is 5V. The anatomy of an RGB LED is illustrated below in Figure 3.

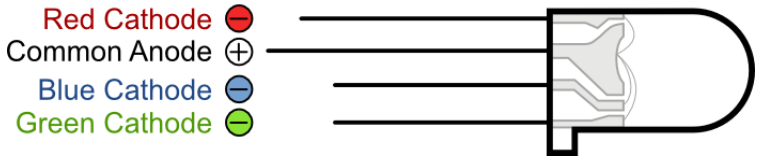


Figure 3: RGB LED Pinout

Using the signal outputs from the microcontroller in pulse-width modulation mode, the RGB LED is controlled. To do so, the microcontroller is set to utilize its timer module in ”Reset/Set Mode”. Therefore, when the timer counts to the value held by CCRx, it will reset the output low. CCRx refers to a capture compare register that is configured to have its own output, set according to the output mode. Moreover, when the timer counts to the value held by CCR0, it will set the output high. This is shown below in Figure 4.

With this output mode, the length of time that the output signal is high may be increased by bringing the CCRx value closer to that of CCR0. Oppositely, the length of time that the output signal is high may be decreased by decreasing the value of CCRx, bringing it further from CCR0. These relationships are the basis for pulse-width modulation; in that, by increasing the CCRx value, not beyond that of CCR0, the duty cycle is increased. By decreasing the CCRx value, the duty cycle is decreased.

4 Implementation

An overview of the implementation is shown below, with specifics being omitted for abstraction purposes, in Figure 5.

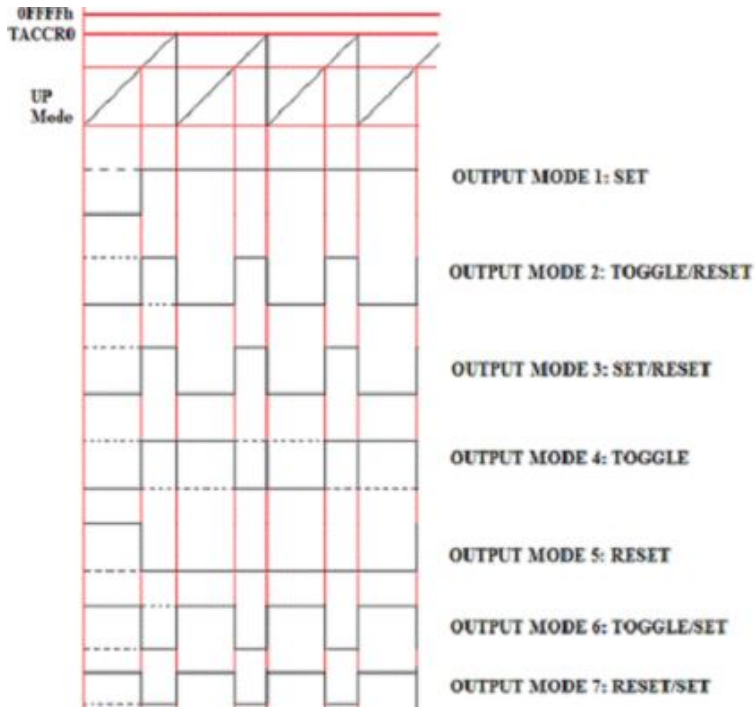


Figure 4: Output Modes; Utilizes Output Mode 7

4.1 UART Configuration

To begin, the UART-echo program from the start-up package was used to configure the UART communication module and its pins. The UART configuration remains the same and the syntax for the UART interrupt exists in its original state. On the MSP430FR6989, P2.0 is configured as the UART transmitting pin and P2.1 as the UART receiving pin. These pins are configured as directed in the datasheet.

4.2 Timer Module Configuration

As it is necessary to configure the timer modules for hardware pulse-width modulation, a timer initialization function is used to set each CCRx register. As the RGB values that are received will be interpreted to be 8-bits, CCR0 is set to 255, the maximum value expressible by one byte. Next, CCR4, CCR5 and CCR6 are initialized to 0. This turns off all of the colors completely. The Timer B module is configured to use the SMCLK in up mode. Lastly, the output mode is set to ”Reset/Set” on each of the capture/compare registers that are used.

The output pins, listed in the data sheet, that can be used for each of the capture/compare registers (CCR4-6) are then configured. The output from TB0.4, or from CCR4,

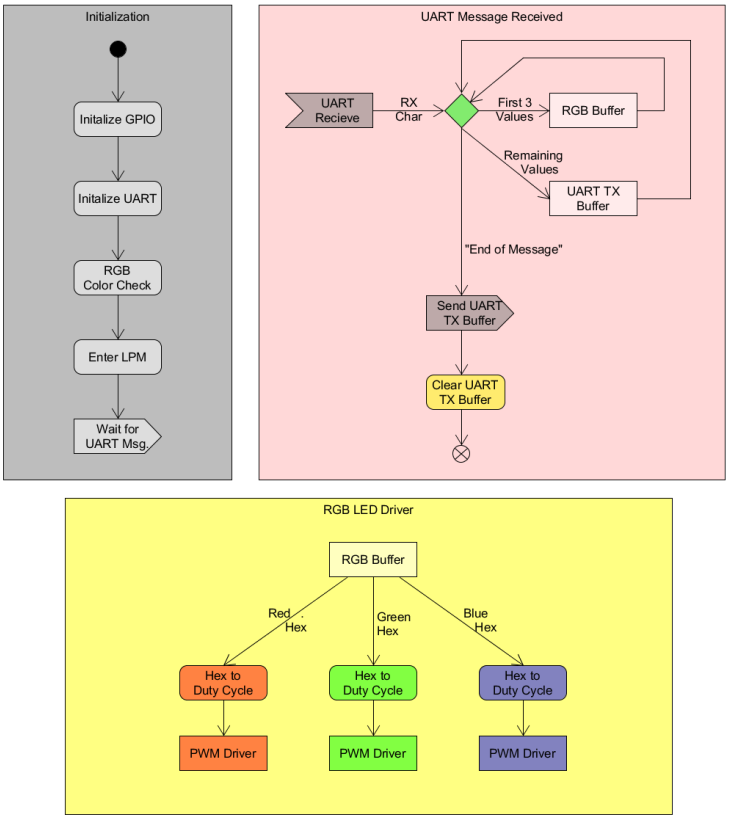


Figure 5: Implementation Overview

is set to P2.5. The outputs from TB0.5 and TB0.6 are then configured on P2.6 and P2.7, respectively. To configure the pins, P2SEL0 is logical OR'd with BIT5 + BIT6 + BIT7. Additionally, P2SEL1 must perform a logical AND with (BIT5 + BIT6 + BIT7). This sets P2.5-P2.7 to use the primary peripheral module. These configuration options are listed in the MSP430FR6989 data sheet and its accompanying Family User Guide.

4.3 Pulse-Width Modulation

The existing UART interrupt is then altered to provide the correct duty cycle to each of the output pins. A counter variable is utilized to identify which byte is being processed at each interrupt. This allows the microcontroller to identify each byte as the length of the string, the red byte, green byte, etc. When the counter variable is equal to 0, the microcontroller transmits the difference of the incoming byte and three. This value is the updated length of the string, after the microcontroller uses three of the bytes for

its own RGB values.

If the counter is equal to 1, CCR4, the register associated with the red pin of the LED, is set to 255 - UCA0RXBUF. UCA0RXBUF is the hex byte that is being processed. The capture/compare register is set to this value, rather than just the incoming hex value, because the LED used is a common-anode RGB. This means that when the signal being sent to each of the color pins is high, the color is off and vice versa. By setting the capture/compare register to the difference of 255 and the incoming hex byte, the duty cycle is set correctly to change the intensity of the color accordingly. This process is repeated for CCR5 (green) and CCR6 (blue).

4.4 Transmission to the Succeeding Node

The counter is incremented until the end-of-string marker is received. When the counter variable reaches 4, it begins to transmit the string. This is how the microcontroller sends the string to the next RGB node. The counter is reset to 0 once the end-of-string marker is received, allowing the microcontroller to stage for another incoming string.

4.5 Hardware

With the MSP430FR6989 programmed to generate its output signals on P2.5(R), P2.6(G), and P2.7(B), the overall functionality is complete after the microcontroller is connected to the RGB LED. The 5V pin of the microcontroller is connected to the common-anode of the LED. Then P2.5 is connected to the red pin of the LED through a 150Ω resistor. Next, P2.6 and P2.7 are connected to green and blue through a 125Ω resistor and a 75Ω resistor, respectively. These resistor values come from the following calculations, where 20mA is the approximate forward current and V_f is the typical forward voltage associated with the LED color. Using these resistor values will adjust for the differences in the various forward voltages, ensuring that the intensity associated with each duty cycle is approximately equal over each color.

$$R_{red} = \frac{V_i - V_f}{20\text{mA}} = \frac{5V - 2V}{20\text{mA}} = 150\Omega$$

$$R_{green} = \frac{V_i - V_f}{20\text{mA}} = \frac{5V - 2.5V}{20\text{mA}} = 125\Omega$$

$$R_{blue} = \frac{V_i - V_f}{20\text{mA}} = \frac{5V - 3.5V}{20\text{mA}} = 75\Omega$$

5 Discussion/Conclusions

Several MSP430 boards were tested before arriving at the FR6989. It was concluded that the FR6989 was best suited for this application because of its Timer B and eUSCI

modules, and also the LCD screen.

The coding itself was deceptively simple. The example UART code from Lab 1: Introduction to C, Git, and the MSP430, while flawed, contained a large portion of the necessary code. It initialized the correct registers for GPIO and UART, and instantiated a switch statement to multiplex between UART interrupt flags. The only remaining implementation done in this lab set up the correct Timer modules – assigning CCRs to specific UART pins – and used another switch statement to assign CCR values to the UART transmit register.

Essentially, the UART example code was corrected and modified to control PWM. This was trivial, yet much effort and time was spent debugging the program. It was not known at the time that the USB's low baud rate was the issue. After bridging the connection with a UART cable instead of the USB cable, the program worked as intended.
Sometimes, opting for the cheapest (or least costly) option is not always the best case. While the F5529 and G2553 are certainly less expensive than the FR6989, the FRAM model has more compelling features. A USB is certainly a feasible method of serial communication, but time and energy wasted trying to achieve functionality. In the end, opting for the dedicated UART cable was the best solution.

With additional time, the FR6989's LCD screen could have been used as another layer of human-machine interfacing. It would allow the user to visualize the bytes being received by the eUSCI.

UART is a very useful protocol for interfacing with microprocessors. This extensively broadens the number of applications possible with the MSP430 family. Controlling the status of an LED has been simplified. Future work will look into and analyze the effectiveness of other protocols, such as I^2C.

References

[1] Texas Instruments, "MSP430FR698x, MSP430FR598x Mixed-Signal Microcontroller" MSP430FR6989 datasheet, Jun. 2014 [Revised Mar. 2017].

[2] Thanigai, P. (2017). Migrating from the USCI Module to the eUSCI Module. [online] Texas Instruments, p.2. Available at: `http://www.ti.com/lit/an/slaa522a/slaa522a.pdf` [Accessed 16 Oct. 2017].

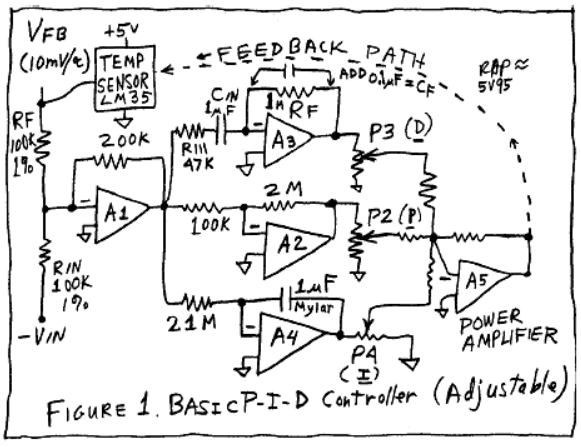
A UART Example Code

```
/* Copyright (c) 2014, Texas Instruments Incorporated
 * All rights reserved.
 */
```

```
#include <msp430.h>

int main(void)
{
  WDTCTL = WDTPW | WDTHOLD;           // Stop Watchdog

  // Configure GPIO
  P2SEL0 |= BIT1;                     // USCI_A0 UART operation
  P2SEL1 &= ~BIT1;

  P4SEL0 | BIT3;
  P4SEL1 &= ~BIT3;

  // Disable the GPIO power-on default high-impedance mode to
     activate
  // previously configured port settings
  PM5CTL0 &= ~LOCKLPM5;

  // Startup clock system with max DCO setting ~8MHz
  CSCTL0_H = CSKEY >> 8;                  // Unlock clock
     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

  // Configure USCI_A0 for UART mode
  UCA0CTLW0 = UCSWRST;                    // Put eUSCI in reset
  UCA0CTLW0 |= UCSSEL__SMCLK;             // CLK = SMCLK
  // Baud Rate calculation
  // 8000000/(16*9600) = 52.083
  // Fractional portion = 0.083
  // User's Guide Table 21-4: UCBRSx = 0x04
  // UCBRFx = int ( (52.083-52)*16) = 1
  UCA0BR0 = 52;                           // 8000000/16/9600
  UCA0BR1 = 0x00;
  UCA0MCTLW |= UCOS16 | UCBRF_1 | 0x4900;
  UCA0CTLW0 &= ~(UCSWRST + UCSYNC);       // Initialize eUSCI
  UCA0IE |= UCRXIE;                       // Enable USCI_A0 RX
     interrupt

  __bis_SR_register(LPM3_bits | GIE);     // Enter LPM3,
     interrupts enabled
  __no_operation();                       // For debugger
}
```

```
#if defined(__TI_COMPILER_VERSION__) || defined(
    __IAR_SYSTEMS_ICC__)
#pragma vector=USCI_A0_VECTOR
__interrupt void USCI_A0_ISR(void)
#elif defined(__GNUC__)
void __attribute__ ((interrupt(USCI_A0_VECTOR))) USCI_A0_ISR (
    void)
#else
#error Compiler not supported!
#endif
{
  switch(__even_in_range(UCA0IV, USCI_UART_UCTXCPTIFG))
  {
    case USCI_NONE: break;
    case USCI_UART_UCRXIFG:
      while(!(UCA0IFG&UCTXIFG));
      UCA0TXBUF = UCA0RXBUF;
      __no_operation();
      break;
    case USCI_UART_UCTXIFG: break;
    case USCI_UART_UCSTTIFG: break;
    case USCI_UART_UCTXCPTIFG: break;
  }
}
```

B Bob Pease



Supplement to Electronic Design August 4,1997

Figure 6: Bob Pease