# Milestone 1: Communicating with Will Byers

*Tiernan Cuesta and Kevin Purcell*
Rowan University

October 22, 2018

## 1 Design Overview

The purpose of the experiment was to generate an RGB node to communicate with Will Byers in the ”upside-down.” Through the use of UART and three individual pulse width modulation signals, the objective was to control the color of the RGB LED by sending packets of bytes to it over serial communication. Multiple MSP430 boards can be connected together, cascading them in any order to send signals down the line of the MSP430 boards to control each individual RGB LED.

### 1.1 Design Features

The design features for the MSP430F5529 launchpad kit RGB LED circuit are as follows:

- PWM
- Serial Communication
- UART, Transfer and Receiving
- RGB LED modulation

### 1.2 Featured Applications

Listed below are some applications that this project could support.

- Christmas Lights
- Party Lights
- Warning Indicator
- Traffic Signals
- Sensor Indicator

## 1.3 Design Resources

Below is a quick link to the code used to complete the objective.

`https://github.com/RU09342-F18/milestone-1-scrum-run/blob/master/Milestone_StrangerThings/LABMS_StrangerThings_F5529/main.c`

## 1.4 Block Diagram

A top-level diagram of this design can be seen in Fig.1. The packets being sent to RX pin are bytes of data that determine the duty cycle of the RGB LED. The packets from TX are the remaining bytes of data then get transferred to the next node, if there is one, if not then the data is returned to the Realterm interface and can be read there.

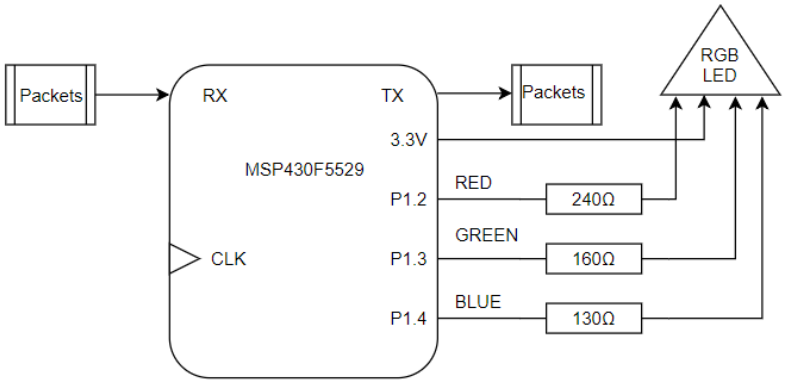


Figure 1: A single LED node using F5529.

### 1.5 Board Image

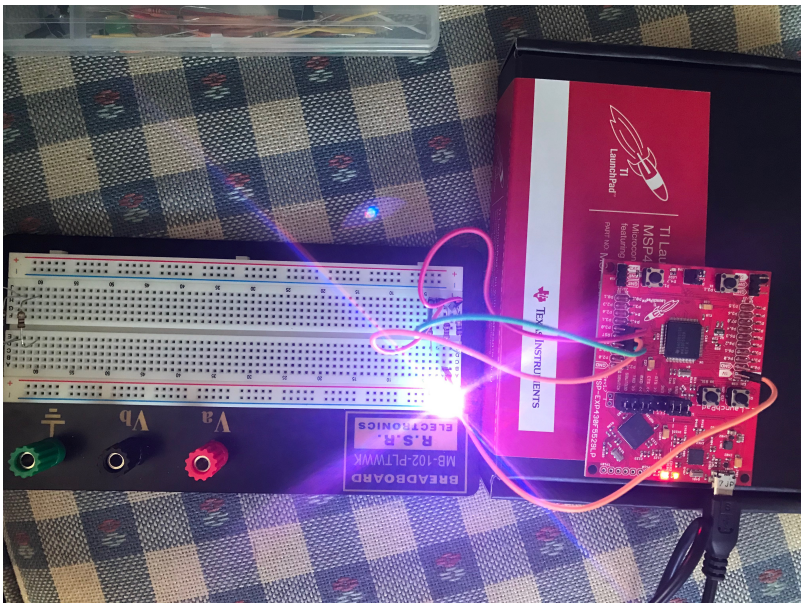

Figure 2: Bread board image of the RGB LED, displaying white.

## 2 Key System Specifications

The parameters used here were sent

| PARAMETERS | SPECIFICATIONS | DETAILS |
|---|---|---|
| BYTE0 | 0x00 to 0xff | Total bytes N being sent |
| BYTE1-(N-2) | 0x00 to 0xff | Brightness of each color for RGB |
| RETURN BYTE | 0xXX | Number of XX packets returned by UART |

Table 1: Various parameter types required by the system

## 3 System Description

In order to properly solve the problem of being able to control the output of a RGB LED, the MSP430F5529 mircocontroler was used. The pin for red was mapped to pin 1.2. The pin for green was mapped to pin 1.3. The pin for blue was mapped to pin 1.4. With the system being powered by 3.3 volts.

## 3.1 Detailed Block Diagram

Below in Fig.3 is a more detailed diagram of the circuitry required. It has the voltages before each diode pin and the current in each loop.

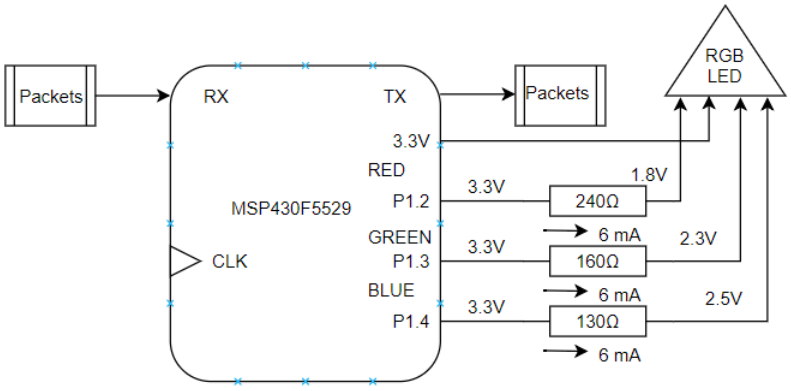


Figure 3: Voltage and current requirements for the design circuit.

## 3.2 Highlighted Devices

This just needs to be a bulleted list of what parts you used (not including passive components) with just a quick blurb of what it is doing in your system.

- MSP430F5529: The microprocessor contains the setup for the LED, PWM, and UART for each of the three LED colors
- RGB LED: Common anode

## 3.3 Device/IC 1

Pulse width Modulation:

Pulse width modulation is used to control the duty cycles for the red, blue, and green diodes to produce specific colors and brightness. Having the MSP430F5529 allows us to only need to implement one clock since each clock contains enough CCRs to implement the three different colors and support the hardware PWM. The clock chosen was the SMCLK in up mode with an input of divider of 1. Each CCR is assigned to 1 of the 3 LEDs in the RGB LED. For example, CCR1 is mapped to the red LED pin.

```
// TimerA assignments to set desired Pulse Width Modulation functionality
void PWM_Setup(void)
{
    TA0CTL = TASSEL_2 + MC_1 + ID_0 + TACLR;   // Sets timer0 to smclk and mode control up
    TA0CCR0 = 0xFF;             // Sets CCR0 as max 255 to cut-off packets to 8 bytes
    TA0CCR1 = 0x00;             // Initializes CCR1 as 0 to receive red LED duty cycle
    TA0CCR2 = 0x00;             // Initializes CCR2 as 0 to receive green LED duty cycle
    TA0CCR3 = 0x00;             // Initializes CCR3 as 0 to receive blue LED duty cycle

    // Sets all three control registers to outmode 3, set/reset mode. This allows for each LED to have its own duty cycle with respect to CCR0
    TA0CCTL1 |= OUTMOD_3;
    TA0CCTL2 |= OUTMOD_3;
    TA0CCTL3 |= OUTMOD_3;
}
```

Figure 4: PWM Setup.

## 3.4 Device/IC 2

UART:

The UART design is mainly based of the code given to us in Lab0. In this instance we re-purposed the code given to us by mainly readjusting the baud rate and modulation that was in the sample code. The baud rate was adjusted to match the speed at which data is transferred on other boards. 9600 Baud rate is translated to 9600 bits per second transferred.

```
38// Various UART protocols to enable receiving and transfer ports to communicate between nodes
39void UART_Setup(void)
40{
41    P4SEL |= BIT4;              // UART TX
42    P4SEL |= BIT5;              // UART RX
43    P3SEL |= BIT3;              // Enables RX for P3.3
44    P3SEL |= BIT4;              // Enables TX for P3.4
45
46    UCA1CTL1 |= UCSWRST;        // Clears the UART control register 1
47    UCA1CTL1 |= UCSSEL_2;       // Sets SMCLK
48    UCA1BR0 = 104;              // For baud rate of 9600
49    UCA1BR1 = 0;                // For baud rate of 9600
50
51    UCA1MCTL |= UCBRS_1;        // Enables modulation control register 1
52    UCA1MCTL |= UCBRF_0;        // Enables modulation control register 0
53    UCA1CTL1 &= ~UCSWRST;       // Enables the UART control register 1
54    UCA1IE |= UCRXIE;           // Enables the UART RX interrupt
55    UCA1IFG &= ~UCRXIFG;        // Clears the UART interrupt flag
56}
57
```

Figure 5: UART Setup.

## 3.5 Device/IC 3

State Machine:

Within the interrupt we establish a switch statement that would control the behavior of the program for each individual case of HEX codes that are received. The first thing received by the processor is the byte length which is handled in the first state. In the first state we take the byte length and subtract 3 than push it along. The reason no operation is implemented in this case is to prevent the processor from missing data while it is sending data. The second to fourth state we have the three different LEDs being handled by the incoming bytes. In the final state which transmits the bytes for the boards down the line.

```
// UART interrupt vector protocol
#pragma vector = USCI_A1_VECTOR
__interrupt void USCI_A1_ISR(void)
{
        switch(bytecount)
        {
        case 0:
            total_bytes = UCA1RXBUF;        // Total byte length
            break;
        case 1:
            TA0CCR1 = UCA1RXBUF;            // Transfers desired duty cycle, or brightness, to the red LED P1.2
            break;
        case 2:
            TA0CCR2 = UCA1RXBUF;            // Transfers desired duty cycle, or brightness, to the green LED P1.3
            break;
        case 3:
            TA0CCR3 = UCA1RXBUF;            // Transfers desired duty cycle, or brightness, to the blue LED P1.4
            while(!(UCA1IFG & UCTXIFG));    // Checks to see if the USCI_A1 TX buffer is ready
            UCA1TXBUF = total_bytes - 3;    // Sends new number of total bytes to next node
            break;
        // Protocol for sending the remaining packets down to the cascading nodes
        default:
            if(bytecount > total_bytes)
            {
                bytecount = -1;
                total_bytes = 0;
            }
            else
            {
                while(!(UCA1IFG & UCTXIFG));// Checks to see if the USCI_A1 TX buffer is ready
                UCA1TXBUF = UCA1RXBUF;      // Transmits bytes to next node
            }
            break;
        }
        bytecount++;                        // Increments bytecount to initialize the next CCRX transfer
}
```

Figure 6: State machine.

# 4 SYSTEM DESIGN THEORY

The overall operation of the system is controlled between the PWM, UART, and TimerA interrupts driven by the F5529. Where the pulse width modulation is used to control the duty cycles for the red, blue, and green diodes to produce a specific colors and brightness. The UART design enables the modulation control registers along with the receive and transfer channels. The UART interrupt utilizes a switch-case function that receives the packets from the previous node and transfers the new total number of bytes and the rest of the packets. The packets are sent in a string of bytes via serial communication using Realterm, a program that can interface with the UART protocol.

## 4.1 Design Requirement 1

Current Limiting Resistors:

Since the MSP430F5529 has a maximum output of 6 mA, current limiting resistor values are needed to be calculated in order to have the proper voltage drop across each pin of the RGB LED. In order to calculate such resistances the desired voltage drop across each pin and the maximum current were used.

$$Red(P1.2) : R = \frac{3.3v - 1.8v}{0.006A} = 240Ohms \tag{1}$$

$$Green(P1.3) : R = \frac{3.3v - 2.3v}{0.006A} = 160Ohms \tag{2}$$

$$Blue(P1.4) : R = \frac{3.3v - 2.5v}{0.006A} = 130Ohms \tag{3}$$

### 4.2 Design Requirement 2

Universal Asynchronous Receiver/Transmitter (UART):

UART stands for Universal Asynchronous Receiver/Transmitter. Its not a communication protocol, but a physical circuit in a microcontroller, or a stand-alone IC. A UARTs main purpose is to transmit and receive serial data. Since the use of UART was mandatory to send packets of infomation from realterm to the MSP430F5529. These packets consisted of bytes which communicated to the MSP430F5529 what color will be shown. UART is responsible for the mapping of the receive and transfer pins, in addition to enabling the modulation control registers. The UART function is needed to also controls the brightness of the RGB LED via duty cycle.

## 5 Getting Started/How to use the device

All that is required to build the circuit is in Fig.1. Next flash the program found in design resources onto the board using code composer, and send packets of bytes to the system via Putty or Realterm. In this practice, Realterm is responsible for interfacing with the UART subsystem.

### 5.1 Building the Circuit

To build the circuit a common anode RGB LED, 3 1/4W resistors, some jumper cables, and the MSP430F5529 launchpad board are required parts. Build the circuit in Fig.1 using the E-24 resistor values on the diagram per the system design requirements.

### 5.2 Debugging and Realterm

Make a new ccs project, make sure the MSP430F5529 is the selected device of the project. Upload the code in the design resource section and run the debug to make sure there's no transfer errors and to flash it to the chip. After debugging the program and flashing it to the F5529 chip Realterm is ready to be setup. First, open the device manager in windows, drop the Ports menu down and the UART comm port should be read there. This is important for setting up Realterm. Under the 'Ports' menu change the port to whichever the comm port named in device manager. Under 'Display' change display from Ascii to Hexspace and check box 'Half Duplex.' Now, under the 'Send' menu the type in a string of packets, for example, a packet of bytes like 0x03 0xff 0xff 0xff would turn the LED white at it's brightest setting.

### 5.3 Hierarchy Chart

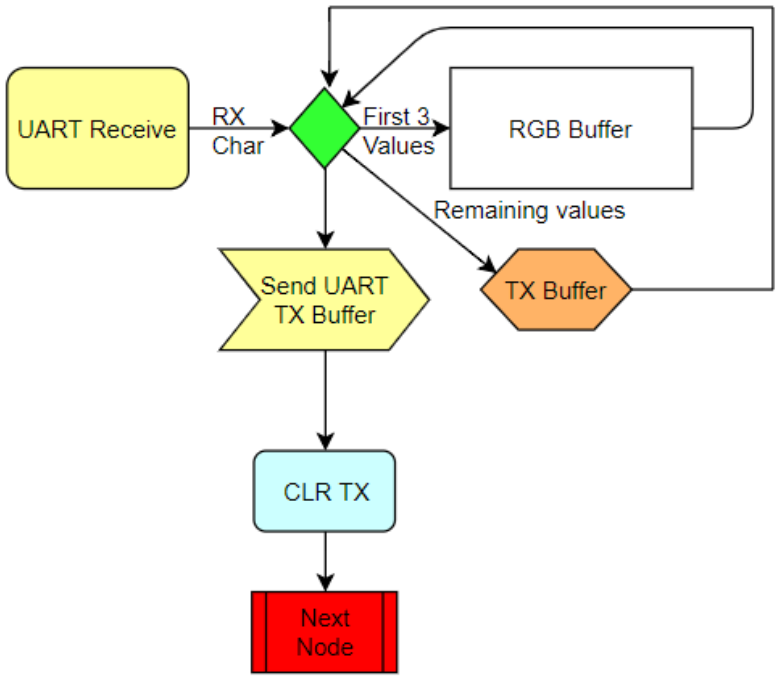


Figure 7: Hierarchy flow chart of UART protocol

## 6 Test Setup

In order to properly set up the device for testing, the off board circuit is needed. consisting of the RGB LED and the three current limiting resistors. Using the MSP430F5529, connect the red pin of the RGB LED to pin 1.2, connect the green pin of the RGB LED to pin 1.3, and connect the blue pin of the RGB LED to pin 1.4. The RGB LED requires 3.3 volts of power, connected the power pin of the RGB LED to the 3.3 volt pin on the MSP430F5529.

Connect the MSP430F5529 and debug and run the program from Code Composer and open up Realterm. Mkae sure the correct com and buad rate is selected and send the following example packet: 0x03 0xff 0x00 0x00, turns the LED red.

## 6.1 Test Data

One single microprocessor is able to receive up to three bytes from a packet.

- 0x03 0xff 0x00 0x00 will turn the RGB LED red
- 0x03 0x00 0xff 0x00 will turn the RGB LED green
- 0x03 0x00 0x00 0xff will turn the RGB LED blue
- 0x03 0xff 0x00 0xff will turn the RGB LED purple
- 0x03 0xff 0xff 0xff will turn the RGB LED white.

If the packet is longer than 3 bytes the MSP430F5529 will return the remaining bytes.

- 0x06 0xff 0x00 0x00 0xff 0xff 0x00 will turn the RGB LED red and will return 03 ff ff 00

## 6.2 Bill of Materials

- MSP430F5529 Texas Instrument Launchpad kit
- Common anode RGB LED
- Prototype board
- Female-to-Male jumper wires
- 240 Ohm 1/4W 5 percent tolerance ceramic resistor
- 160 Ohm 1/4W 5 percent tolerance ceramic resistor
- 130 Ohm 1/4W 5 percent tolerance ceramic resistor
- Code Composer / Realterm