Milestone 1 Application Note
MSP430 UART Stranger Things Project
By: Chris Iapicco and Bryan Regn

1 Overview

This embedded software was designed to be one modular node in a chain of microprocessors that will implement a UART-coordinated color scheme. This chip is designed to be used in a chain, or on its own, and will function properly regardless of where in the chain it is located. The chips need only be connected over UART for the function to work properly, and each node in the chain is essentially acting as a slave, with one master sending commands. The chip used is the MSP430FR5994, located on a launchpad, and all provided electrical characteristics are pulled directly from the data sheet for the chip.

2 MSP430FR5994 Specifications

2.1 Maximum Ratings

	MIN	MAX	UNIT
Voltage applied at DVCC and AVCC pins to V_{SS}	–0.3	4.1	V
Voltage difference between DVCC and AVCC pins[2]		±0.3	V
Voltage applied to any pin [3]	–0.3	V_{CC} + 0.3 V (4.1 V Max)	V
Diode current at any device pin		±2	mA
Storage temperature, T_{stg} [4]	–40	125	°C

2.2 Recommended Operating Conditions

			MIN	NOM	MAX	UNIT
V_{CC}	Supply voltage range applied at all DVCC and AVCC pins[1] [2] [3]		1.8[4]		3.6	V
V_{SS}	Supply voltage applied at all DVSS and AVSS pins.			0		V
T_A	Operating free-air temperature		–40		85	°C
T_J	Operating junction temperature		–40		85	°C
C_{DVCC}	Capacitor value at DVCC[5]		$1_{-20\%}$			µF
f_{SYSTEM}	Processor frequency (maximum MCLK frequency)[6]	No FRAM wait states (NWAITSx = 0)	0		8[7]	MHz
		With FRAM wait states (NWAITSx = 1)[8]	0		16[9]	
f_{ACLK}	Maximum ACLK frequency				50	kHz
f_{SMCLK}	Maximum SMCLK frequency				16[9]	MHz

3 Software Design

3.1 Chain Layout

The design of the software on this chip was taken from the requirements provided, and can be seen in Figures 1 and 2 below. The end design goal is a theoretically endless chain of embedded microprocessors that will take in a UART code, use specific bytes to produce a specific color, and pass on the remaining bytes to the next microprocessor.

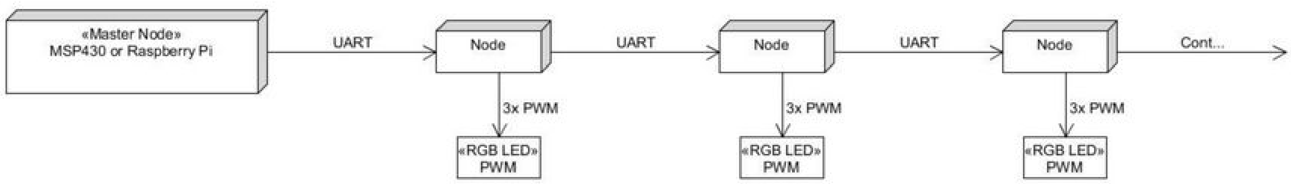

Figure 1. Design layout of the complete chain of RGB nodes that this chip is designed to be a part of.

As it appears in Figure 2. The single node is designed to be modular, to fit into any point of the chain. Each individual node will take in the full UART command, chop off three bytes, and use those bytes as the duty cycle for three different PWM signals corresponding to a red, green, and blue LED to create a unique color. The node will send the remaining bytes down the chain.

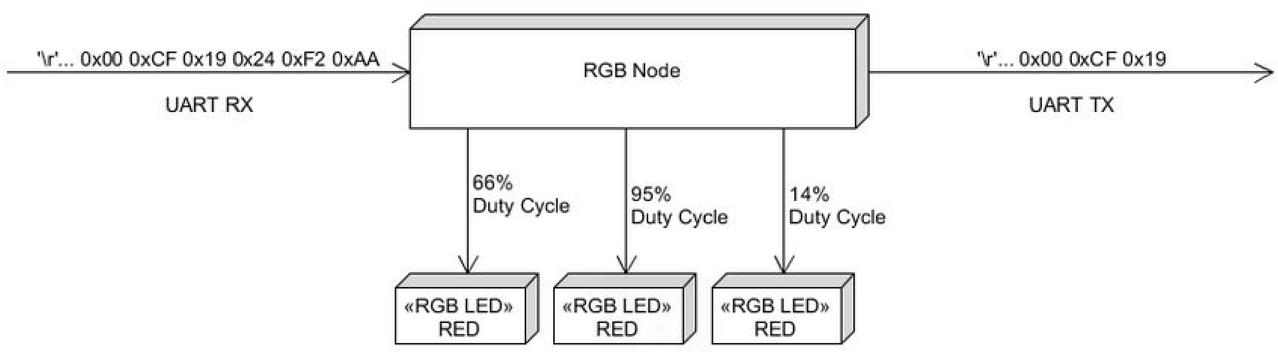

Figure 2. In depth functionality layout of a single node of the RGB chain.

3.2 UART Message Protocol

The UART message protocol was taken directly from the requirements provided. The UART command that comes from the master, and runs down the chain of microprocessors will always have the same format, as seen in Figure 3.

Byte Number	Contents	Example
Byte 0	Number of bytes (N) including this byte	0x50 (80 bytes)
Bytes 1-(N-2)	RGB colors for each node	0xFF (red) 0x00 (green) 0x88 (blue) ...
Byte N-1	End of Message Character	0x0D (carriage return)

Figure 3. Message layout of the UART command that will run through the chain.

3.3 Software Structure

The structure for the software was based off of the requirements given, and can be seen in Figure 4. This chart excludes byte 0, which is handled by subtracting 3 from the hexadecimal value of the character, and adding it to the front of the UART command that is sent to the next microprocessor.

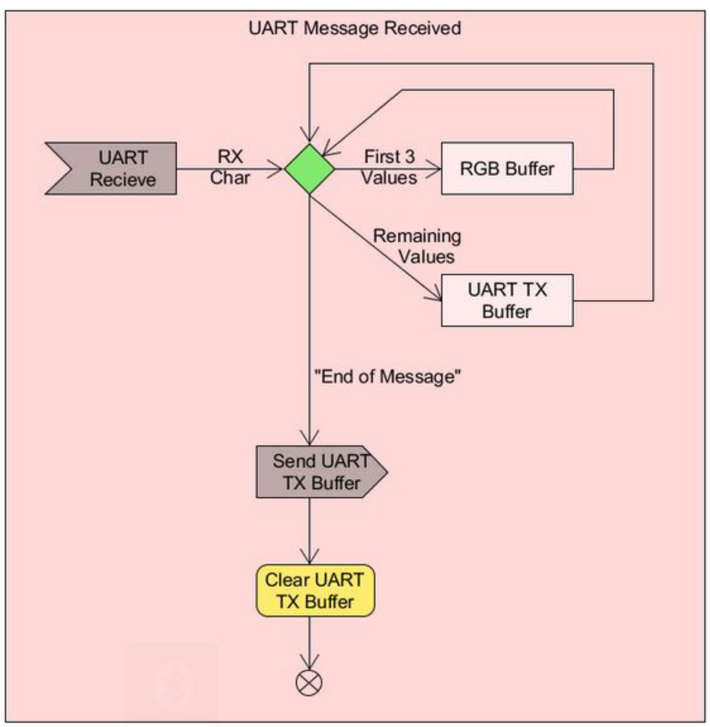

Figure 4. Software structure for the UART message handling.

The beginning of the code, seen in Figure 5, is structured to initialize the TimerB0 module to act as the PWM for the red, green, and blue LEDs. The period of the PWM was set to 255 to be able to use the value transmitted to the chip through UART as the duty cycle without any calculations or modulations. The PWMs were set to reset/set again in order to avoid having to modulate the incoming UART code. The outputs for the PWM modules were set to pins that could be accessed directly on the launchpad board.

```
TB0CTL|=BIT1; //Enable TimerB0 capture compare intecupt
TB0CCTL1=OUTMOD_7; //Red LED reset/set mode
TB0CCTL2=OUTMOD_7;//Green LED reset/set mode
TB0CCTL3=OUTMOD_7;//Blue LED reset/set mode
TB0CTL = TASSEL_2 + MC_1 + TACLR; // SMCLK, upmode

TB0CCR0=255; //Period of PWM
TB0CCR1=255; //Red LED duty cycle
TB0CCR2=255; //Green LED duty cycle
TB0CCR3=255; //Blue LED duty cycle

P1DIR|=(BIT5+BIT4); //P1.4, P1.5 set to output TimerB0 capture compare outputs 1 and 2
P1SEL0|=(BIT5+BIT4);//P1.4, P1.5 set to output TimerB0 capture compare outputs 1 and 2
P3DIR|=BIT4;//P3.4 set to output TimerB0 capture compare output 3
P3SEL0|=BIT4//P3.4 set to output TimerB0 capture compare output 3

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

Figure 5. Code for the configuration of TimerB0 and the corresponding RGB PWM outputs

Much of the code for initializing the UART module and the baud rate was taken from a Texas instruments provided example code, written in October 2015 by William Goh which demonstrated how to echo incoming UART code. This code initialized the UART with a baud rate of 9600 which was a requirement for compatibility with the line of RGB nodes. The majority of this code, implemented in the RGB node project can be seen in Figure 6. The chip is put into low power mode 3 to save power in between interrupts.

```
// Configure UART pins
P2SEL0 &= ~(BIT0 | BIT1);//
P2SEL1 |= BIT0 | BIT1;                  // USCI_A0 UART operation


// Startup clock system with max DCO setting ~8MHz
CSCTL0_H = CSKEY_H;                     // Unlock CS 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
UCA0BRW = 52;                           // 8000000/16/9600
UCA0MCTLW |= UCOS16 | UCBRF_1 | 0x4900;
UCA0CTLW0 &= ~UCSWRST;                  // Initialize eUSCI
UCA0IE |= UCRXIE;                       // Enable USCI_A0 RX interrupt

__bis_SR_register(LPM3_bits | GIE);     // Enter LPM3, interrupts enabled
```

Figure 6. Uart configuration code.

Figure 7 shows the interrupt vector for the UART module. This portion of the code functions by differentiating between the incoming UART bytes, and handling each differently. The code detects when the first byte should be incoming, and saves this character to two registers: one to save how many bytes should be received, and one to decrement each time a character is handled. When the decrementing register is zero, this means that all bytes in the previous command have been handled, and the next incoming character is the first in a new command. The next three characters, the red, green, and blue LED duty cycles respectively, are detected by comparing the decrementing register to the expected bytes in the command. In the handling of the blue LED's duty cycle, the expected bytes to be sent are sent down the UART line. For the rest of the UART bytes received, until the command is finished, the byte that is received is immediately sent down the line.

```
{
        if (decrement==0) //If first byte recieved
     {
       total_bytes = UCA0RXBUF; //Total_bytes is updated with the character recieved
       decrement = total_bytes; //The counting register is updated with the expected # of bytes
       decrement--; //The counting register is decremented
     }
    else if (decrement== (total_bytes-1)) //If Red LED duty cycle byte
    {
      TB0CCR1=UCA0RXBUF;//Red LED duty cycle is updated with byte received
      decrement--;//Counting register is decremented
    }
  else if (decrement==(total_bytes-2)) //If Green LED duty cycle byte
    {
             TB0CCR2=UCA0RXBUF; //Green LED duty cycle updated with byte received
             decrement--;//Counting register is decremented
  }
  else if (decrement==(total_bytes-3)) //If Blue LED duty cycle byte
  {
    while (!(UCA0IFG&UCTXIFG));//If the TXBUF is ready to send move on
    TB0CCR3=UCA0RXBUF;//Blue LED duty cycle updated with byte received
    UCA0TXBUF=(total_bytes-3); //Send updated amount of bytes that will be in UART command
    decrement--;//Counting register is decremented
  }

    else //All bytes after the fourth byte
  {
    while (!(UCA0IFG&UCTXIFG));   //If the TXBUF is ready to send move on
    UCA0TXBUF=UCA0RXBUF; //Send byte received
    decrement--; //Counting register is decremented
  }
}
```

Figure 7. UART interrupt service routine

3.4 MSP430FR5994

The MSP430FR5994 suits the needs of this project perfectly. The TimerB0 module has enough capture compare modules to handle the PWM of three LEDs simultaneously. Additionally, the UART module is powerful enough to handle the requirements. Furthermore, the MSP430FR5994 had previously created, open source code available for use which initialized the UART to the exact needs of this design, reducing engineering hours.

This chip is able to handle a deep low power mode while handling both UART interrupts, and PWM signals, making this chip an excellent choice for this implementation.

4 Pin Layout

Table 1 displays the ports that are being used for the PWM outputs on the board. The corresponding figure, Figure 8, taken from the Launchpad user guide for the MSP430FR5994 shows where to hook up to these ports, as well as VCC and GND.

Table 1. Ports used to output the LED PWM signals

Port	Function
1.4	Red LED output
1.5	Green LED output
3.4	Blue LED output

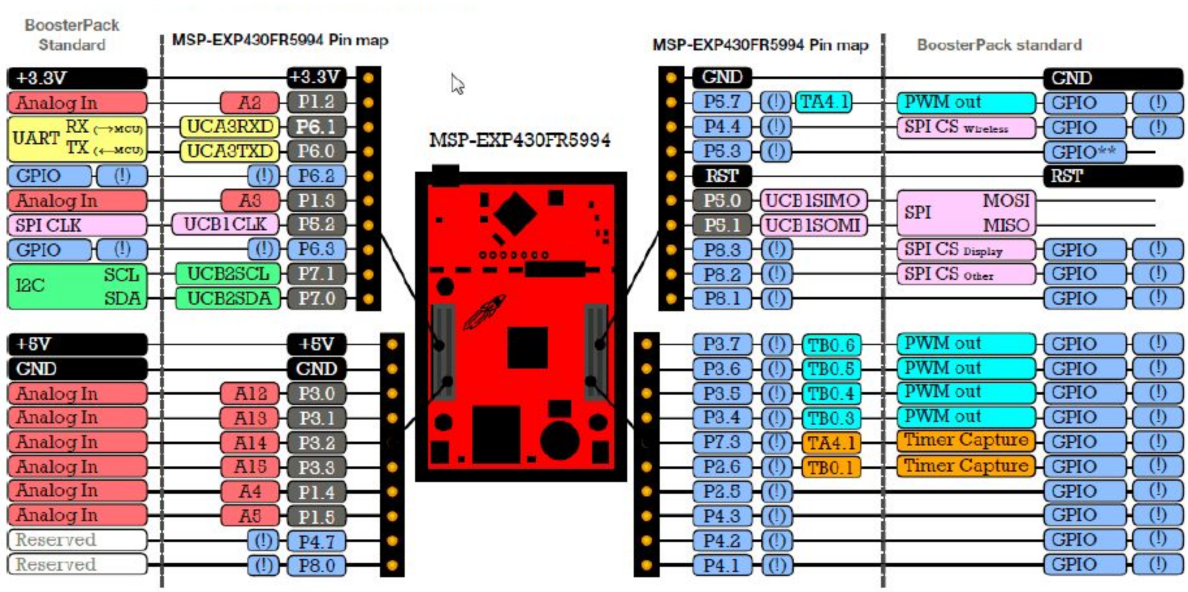

Figure 8. Launchpad guide showing which headers to connect the LEDs, VCC, and GND

The Uart connection should be taken from jumper J101, the jumper connecting the emulator to the launchpad. The pins labeled “RXD” and “TXD” are the UART receiving and transmitting lines respectively and can be accessed by taking off the jumpers, and using a female connector.

References

Node requirements:
https://github.com/RU09342/milestone-1-communicating-with-will-byers-team-316
William Goh UART example code:
https://github.com/RU09342/lab-1-intro-to-git-c-and-msp430-iapiccoc9/tree/master/Example%20Code/MSP430FR5994
MSP430FR5994 datasheet: http://www.ti.com/lit/ds/symlink/msp430fr5992.pdf
MSP430FR5994 Launchpad user guide: http://www.ti.com/lit/ds/symlink/msp430fr5992.pdf