Introduction to Embedded Systems - Milestone 1

Matt Mammarelli and Max Tensfeldt
Rowan University

October 16, 2017

1 Abstract

The objective is to control the color of an RGB LED through the use of UART and three individual PWM signals. Multiple MSP430 boards can be connected together in any order using UART and the code should still work. The MSP430F5529 board was chosen for this implementation.

2 Introduction

All of the boards were considered but the MSP430F5529 came out on top because of the easy and reliable implementation for hardware PWM along with reliable UART communication using the USB serial cable. PWM also known as pulse width modulation is a way to control the amplitude of a signal with the width of the pulses of a square wave. The duty cycle is the ratio of the time the pulse is on over the period of the waveform. The MSP430F5529 supports built in PWM functionality which can be utilized to control the brightness or color of an led. UART is also known as Universal Asynchronous Receive Transmit and is a serial protocol used to transfer data. Using a terminal program like Realterm, serial data can be sent to the MSP430F5529. The MSP430F5529 can also transfer data using the UART protocol. These techniques will be utilized to complete this milestone.

3 Background

There were problems with the other boards that led to our choice to use the MSP430F5529. The MSP430FR2311 only has Timer B, does not have reliable PWM, and only works with UART USB back channel. The MSP430FR5994 doesn't reliably work with hardware PWM and doesn't reliably work with the UART serial cable. The MSP430FR6989 doesn't work at all with either UART USB back channel or serial cable. The MSP430G2553 doesn't have enough CCR registers for 3 PWM. All of these factors led to the decision to use the MSP430F5529.

4 Circuit

The RGB LED was common anode so a circuit needed to be created to support this.

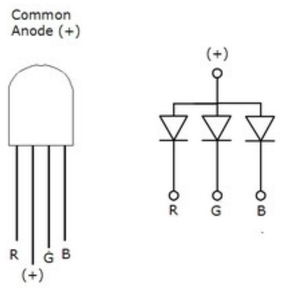


Figure 1: RGB LED Common Anode Pinout

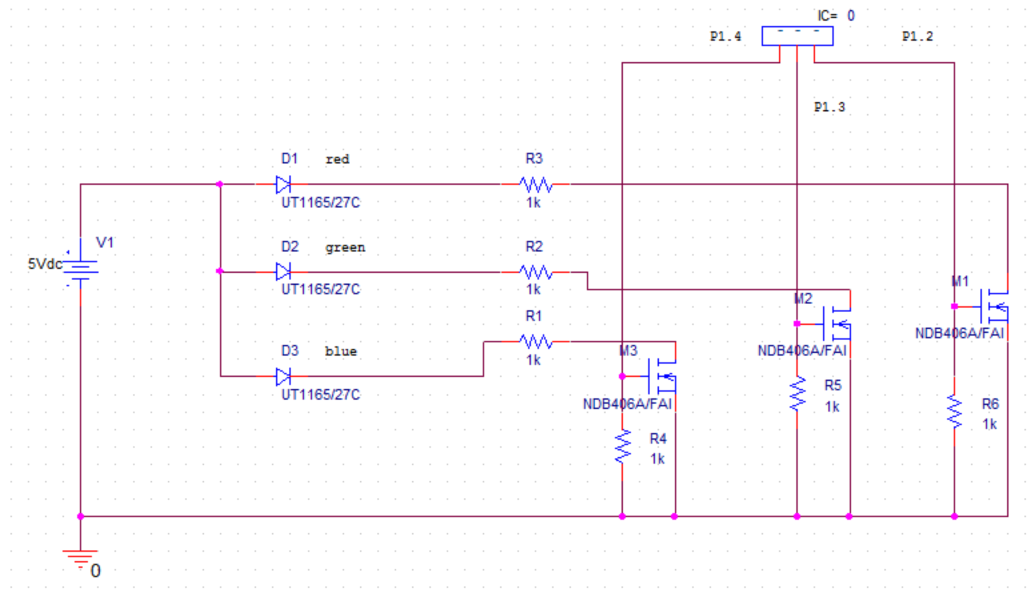


Figure 2: Circuit Schematic

4.1 PWM

Hardware PWM was utilized for the MSP430F5529 for simplicity since this board supports up to seven PWM outputs. We need three PWM signals, one for each color of the led. The common anode means that the longest leg of the led must be hooked up to VCC which I chose to be 5V. Each leg then is connected to a 1K resistor to limit the current running through the nMOS transistors. The nMOS transistors are used as low side switches. This means that the gate is connected to a GPIO pin on the microcontroller which controls whether the led is on or not. 1K pulldown resistors are needed between the gate and ground for each of these transistors so that they aren't floating. The headers on the bottom of the board were used to breakout the pins and power to a breadboard. The pins chosen to control the leds are P1.2 for Red, P1.3 for Green, and P1.4 for Blue.

4.1.1 PWM Code Explanation

The PWM used for this board is hardware based and was made to be approximately 1kHz. Three CCR control registers were created and used the built in reset/set functionality. The 1mHZ SMCLK in Up Mode was also used. This was chosen for simplicity as the TAR register would just count up to each CCR register to control the PWM for each color.

4.2 UART

The UART structure was standardized so that each node in the chain will parse through 8 bytes worth of data to control the PWM on that board as well as what data to send to the next. Using a serial to usb cable, the green was connected to P3.4 and white connected to P3.3. A packet with RGB values will be sent to the next board in the chain while the values that were used for the current board were removed.

Byte 0: Total bytes in package 0x08
Byte 1: Red duty cycle current board
Byte 2: Green duty cycle current board
Byte 3: Blue duty cycle current board
Byte 4: Red duty cycle next board
Byte 5: Green duty cycle next board
Byte 6: Blue duty cycle next board
Byte 7: End of message 0X0D

4.2.1 UART Code Explained

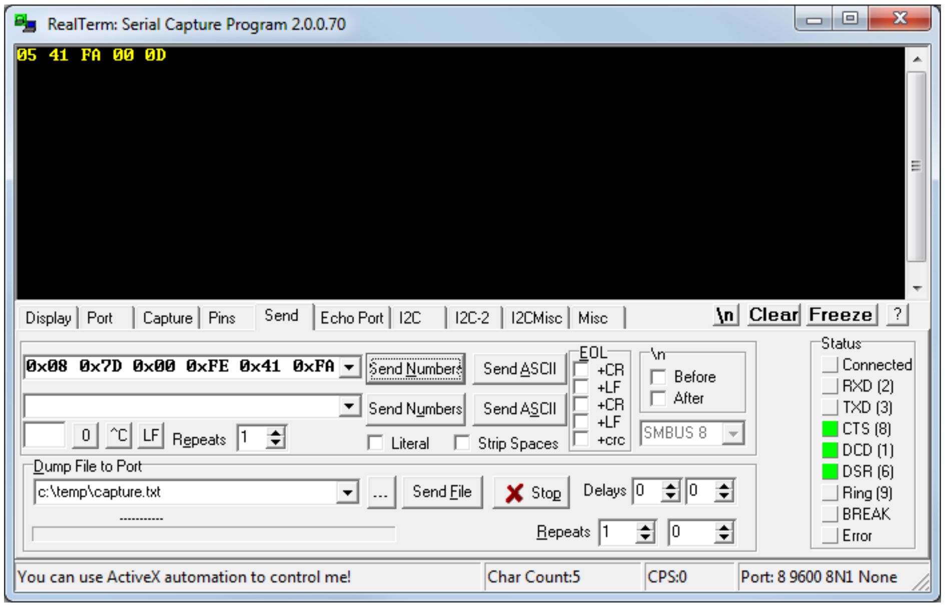
Some of the variables used for the UART control are byteCount, numBytes, and red,green,blue. The byteCount keeps track of the current byte being parsed, numBytes is the total number of bytes in the received packet, and the red, green, and blue variables will hold the received PWM values for the red, green, and blue colors.

Inside the UART Interrupt vector, when the first bit is parsed, numBytes is set to be that byte. Then when byteCount is incremented, it is checked in a switch statement to see whether the red, green, or blue PWM is being received. Each received PWM is multiplied by 4 since we chose the period to be 1024 which would be 256*4. The 256 comes from the range of values that control each PWM led which would be 0 to 255. The first transmission byte is chosen to be numBytes -3 since we are taking out the current rgb values from the packet and only sending the next board rgb values. Finally, when the end of the packet is found to be 0x0D, the byteCount is reset, the transmission is set to 0X0D for the final byte, and the board is ready to receive a new UART signal.

4.2.2 UART Testing

The UART was tested using Realterm where an 8 byte string was sent to the board to turn the led a purple color. The Realterm displayed the data that will be transmitted to the next board which are rgb values encapsulated in a new 5 byte package.

Sent: 0x08 0x7D 0x00 0xFE 0x41 0xFA 0x00 0x0D


RealTerm: Serial Capture Program 2.0.0.70

```
05 41 FA 00 0D
```

0x08 0x7D 0x00 0xFE 0x41 0xFA

Dump File to Port: c:\temp\capture.txt

You can use ActiveX automation to control me! | Char Count:5 | CPS:0 | Port: 8 9600 8N1 None

Figure 3: Realterm

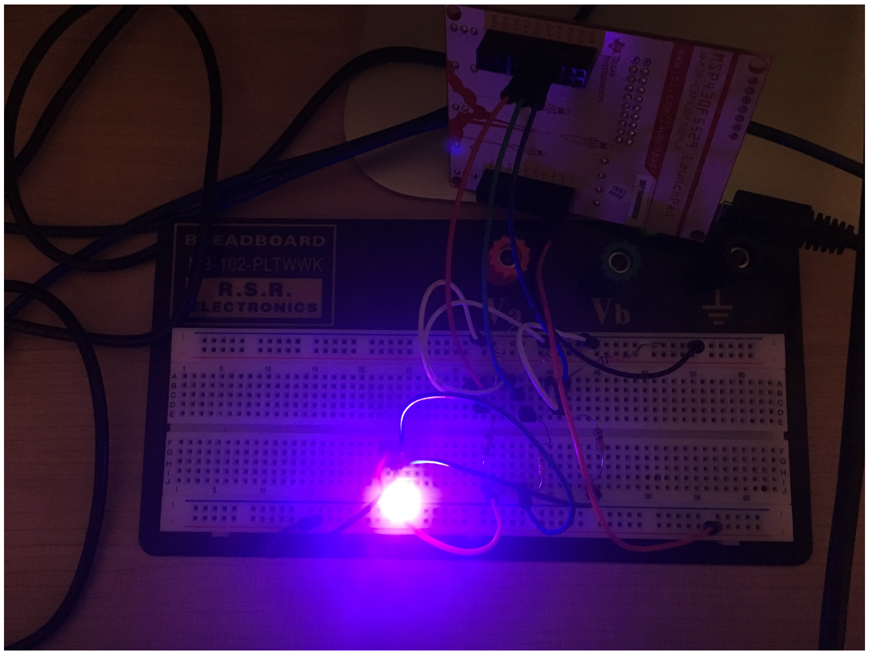


Figure 4: Circuit Led ON

5 Code

The code was broken up into a UART block and a PWM block. The UART is received, is iterated through byte by byte and the data is either received or transmitted based on the protocol defined above.

```
/*
Matt Mammarelli
9/18/17
ECE 09342-2
*/

//MSP430F5529 Milestone
//Takes a string of bits over UART, parses it for the current and next rgb
    values, then transmits the next board rgb
//When incoming is 56 or greater bytes the txd will only output 48 bytes, need
    to reset after first transmit


#include <msp430f5529.h>

int byteCount=0; //the current byte that is iterating through the packet
int numBytes=0; //number of bytes in packet
int red,green,blue=0; //holds UART values for current rgb


void main(void)
{

  //stop watchdog timer
  WDTCTL = WDTPW + WDTHOLD;

  //uart ******************************************************************
  // P3.3, P3.4 transmit/receive
  P3SEL = BIT3+BIT4;
  // Put state machine in reset
  UCA0CTL1 |= UCSWRST;
  // SMCLK
  UCA0CTL1 |= UCSSEL_2;
  // 1MHz 9600 baud
  UCA0BR0 = 6;
  // 1MHz 9600
  UCA0BR1 = 0;
  //sets m control register
  UCA0MCTL = UCBRS_0 + UCBRF_13 + UCOS16;
  //sets control register
  UCA0CTL1 &= ~UCSWRST;
  //enable interrupt
  UCA0IE |= UCRXIE;
  //**********************************************************************


  //rgb pwm ***************************************************************

  // P1.2 , P1.3, P1.4 output
  P1DIR |= BIT2+BIT3+BIT4;

  // P1.2 and P1.3, P1.4 options select GPIO
  P1SEL |= BIT2+BIT3+BIT4;

  // PWM Period about 1khz
  TA0CCR0 = 1024;

  // CCR1 reset/set
  TA0CCTL1 = OUTMOD_7;

  // CCR2 reset/set
  TA0CCTL2 = OUTMOD_7;

  // CCR3 reset/set
  TA0CCTL3 = OUTMOD_7;
  // SMCLK, up mode, clear TAR
  TA0CTL = TASSEL_2 + MC_1 + TACLR;

  //*************************************************************************

  // Low power mode
  __bis_SR_register(LPM0_bits + GIE);

  // For debugger
  __no_operation();
}




//uart interrupt vector
#pragma vector=USCI_A0_VECTOR
__interrupt void USCI_A0_ISR(void)
{
  switch(__even_in_range(UCA0IV,4))
  {
  case 0:break;  // Vector 0 - no interrupt
  case 2:{
      while (!(UCA0IFG&UCTXIFG)); // USCI_A0 TX buffer check
          if(byteCount==0){
                      numBytes = UCA0RXBUF;
                      byteCount++;


                  }
                  //current rgb
                  else if ((byteCount>0 & byteCount <4)){
                      switch(byteCount){
                      case 1:{
                          red = UCA0RXBUF;
                          // CCR1 PWM duty cycle red
                          TA0CCR1 = red * 4;
                          break;
                      }
                      case 2:{
                          green = UCA0RXBUF;
                          // CCR2 PWM duty cycle green
                          TA0CCR2 = green * 4;
                          break;
                      }
                      case 3:{
                          blue = UCA0RXBUF;
                          // CCR3 PWM duty cycle blue
                          TA0CCR3 = blue * 4;
                          //beginning of new transmit message
```

```
                                        UCA0TXBUF = numBytes-3;
                                        break;
                                    }
                                    default:break;


                                    }

                                    byteCount++;

                                }
                                //sending rgb and rest of message
                                else if (byteCount>3 & byteCount <= numBytes-1){
                                    if (byteCount!=numBytes-1){
                                        UCA0TXBUF = UCA0RXBUF;
                                        byteCount++;
                                    }
                                    else{
                                        UCA0TXBUF = 0x0D; //end of new message
                                        byteCount=0;
                                    }

                                }

                    break;

            }

            case 4:break;   // Vector 4 - TXIFG
            default: break;
            }
        }
```