Brian Lopez
Github root directory: github.com/brian4280/Bed_Assign

Date Submitted: **10/12/18**

Task 01:

Youtube Link: https://www.youtube.com/watch?v=q9By8sZAZBw

Modified Code:

- **In main, turn on timer1 and ADC:**

```
int main(void) {
    task0Setup(); // does everything from task0
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1);
    // turn on timer 1 for 0.5 second intervals
    TimerConfigure(TIMER1_BASE, TIMER_CFG_PERIODIC);
    TimerLoadSet(TIMER1_BASE, TIMER_A, 20000000-1);
    IntEnable(INT_TIMER1A);
    TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
    // configure ADC for Sequence 1
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);

    ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE,1,3,ADC_CTL_TS|ADC_CTL_IE|ADC_CTL_END);

    TimerEnable(TIMER1_BASE, TIMER_A); // turn on Timer

    while (1) //let interrupt handler do the UART echo function
    {
        // i
    }
}
```

- **Inside the Timer Interrupt function, turn on ADC and grab the ADC values. Once it's done, convert values to C and F. Once that's done, convert the int value to a string and then send it through UART:**

```
void Timer1IntHandler(void)
{
    uint32_t TVals[4]; // Values for temperature
    uint32_t TempF, TempC, Avg, i;
    char buff[20];

    //clear flags
    TimerIntClear(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
    ADCIntClear(ADC0_BASE, 1);

    //start ADC
    ADCSequenceEnable(ADC0_BASE, 1);
    ADCProcessorTrigger(ADC0_BASE, 1);

    while(!ADCIntStatus(ADC0_BASE, 1, false))
    {
        // poll until ADc is done
    }

    ADCSequenceDataGet(ADC0_BASE, 1, TVals); // get adc values
    Avg = (TVals[0] + TVals[1] + TVals[2] + TVals[3] + 2) / 4;
    TempC = (1475 - ((2475 * Avg)) / 4096) / 10; // convert to C
    TempF = ((TempC * 9) + 160) / 5;  // convert to F

    ltoa(TempF, buff);  // convert int to a string
    i = 0;
    while(buff[i] != '\0')
    {
        UARTCharPut(UART0_BASE, buff[i]);  // send all characters through UART
        i++;
    }
    UARTCharPut(UART0_BASE, '\r');   // put new line and carraige return
    UARTCharPut(UART0_BASE, '\n');
    ADCSequenceDisable(ADC0_BASE, 1);  // turn off ADC
}
```

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

Task 02:

Youtube Link: https://www.youtube.com/watch?v=uT3MOn4XEKE

Modified Code:

- **The first thing I did was create a function that can send a string through UART so that I don't have to constantly do UARTCharPut for each letter:**

```
void sendString(char text[])
{
    int i = 0;
    while(text[i] != '\0')  // sends until NULL is found
    {
        UARTCharPut(UART0_BASE, text[i]);
        i++;
    }
}
```

- **In main, the only thing changed was to enable portF for the LEDs as an output and then ask for a command:**

```
int main(void) {
    task1Setup(); // does everything from task 0 and 1

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF); // enable LED's as output
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3);
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3, 0);

    // send string through UART
    sendString("Enter Command: ");


    while (1) //let interrupt handler do the UART echo function
    {
        //
    }
}
```

- **In the UART interrupt, it will get the char value passed through and check which letter was passed. For R, G, B, and their lowercase versions, it sent back to letter to echo, then turned on/off their respective light.**
- **For 'T' it turns on ADC, gets the temperature values and converts them to C and F. Then it will send everything over using the sendString function.**
- **If an unknown char is sent, then the help prompt will be sent that says what can letters do what.**

```
void UARTIntHandler(void)
{
    uint32_t ui32Status;
    ui32Status = UARTIntStatus(UART0_BASE, true); //get interrupt status
    UARTIntClear(UART0_BASE, ui32Status); //clear the asserted interrupts

    uint32_t Vals[4];
    uint32_t Avg;
    uint32_t TempC, TempF;

    char Cel[10];
    char Far[10];
    char command;
    command = UARTCharGet(UART0_BASE);
    if (command == 'R') { // turn on red LED
        UARTCharPut(UART0_BASE, command);  // echo back letter
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, GPIO_PIN_1); } // turn on light
    else if (command == 'B'){ // same as above
        UARTCharPut(UART0_BASE, command);
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, GPIO_PIN_2); }
    else if (command == 'G') {
        UARTCharPut(UART0_BASE, command);
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, GPIO_PIN_3); }
    else if (command == 'r') {  // turn off red LED
        UARTCharPut(UART0_BASE, command);
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0); } // turn off light
    else if (command == 'b') { // same as above
        UARTCharPut(UART0_BASE, command);
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0); }
    else if (command == 'g') {
        UARTCharPut(UART0_BASE, command);
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0); }
    else if (command == 'T') { // send temperature through UART
        UARTCharPut(UART0_BASE, command); // echo command
        ADCIntClear(ADC0_BASE, 1);  // clear flag
        ADCSequenceEnable(ADC0_BASE, 1);  // enable ADC
        ADCProcessorTrigger(ADC0_BASE, 1);
        while(!ADCIntStatus(ADC0_BASE, 1, false))
        {

        }
        ADCSequenceDataGet(ADC0_BASE, 1, Vals);  // get the 4 values
        Avg = (Vals[0] + Vals[1] + Vals[2] + Vals[3] + 2) / 4;

        // convert to C and F
        TempC = (1475 - ((2475 * Avg)) / 4096) / 10;
        TempF = ((TempC * 9) + 160) / 5;

        // convert int to string
        ltoa(TempF, Far);
        ltoa(TempC, Cel);

        // send statement through UART
        sendString("\r\nTemp = ");
        sendString(Cel);
        sendString("C = ");
        sendString(Far);
        UARTCharPut(UART0_BASE, 'F');
    }
    else { // unknown command, send help
        UARTCharPut(UART0_BASE, command);
        sendString("\r\nR: red, G: green, B: blue, T: temperature\r\n");
    }

    // resend command string
    sendString("\r\nEnter command: ");

}
```

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.