Ali Asadi
Github root directory: https://github.com/aliasadivet/Lab_Submition-.git

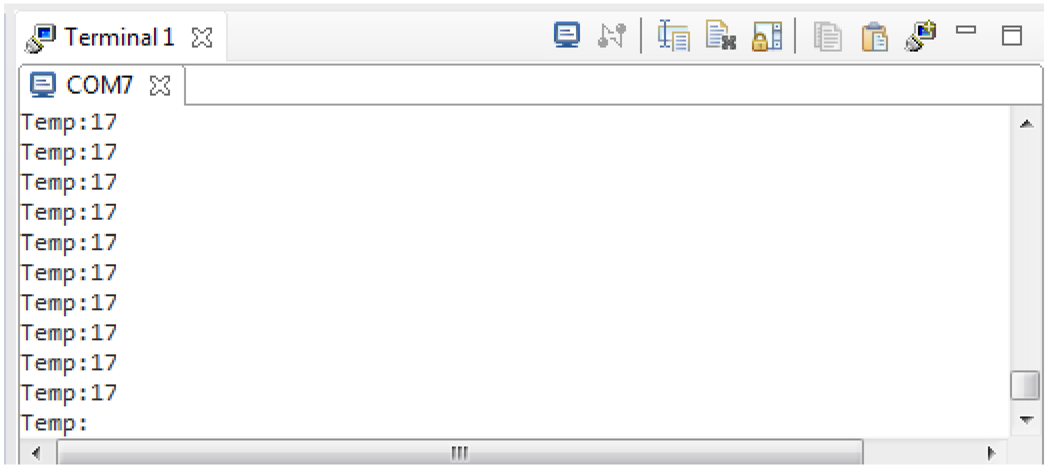
Lab07

Task 1

Modified code:

```
#include <stdint.h>
#include<stdlib.h>
#include <stdbool.h>
#include<string.h>
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"

#include "driverlib/rom.h"
#include "driverlib/gpio.h"
#include "driverlib/timer.h"
#include "driverlib/interrupt.h"
#include "driverlib/adc.h"

void UARTIntHandler(void)
{
 uint32_t ui32Status;
 ui32Status = UARTIntStatus(UART0_BASE, true); //get interrupt status
 UARTIntClear(UART0_BASE, ui32Status); //clear the asserted interrupts
 while(UARTCharsAvail(UART0_BASE)) //loop while there are chars
 {
 UARTCharPutNonBlocking(UART0_BASE, UARTCharGetNonBlocking(UART0_BASE)); //echo
character
 GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, GPIO_PIN_2); //blink LED
 SysCtlDelay(SysCtlClockGet() / (1000 * 3)); //delay ~1 msec
 GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0); //turn off LED
 }
}
char* itoa (uint32_t value, char * buffer)
 {
     char buff[8];
     int i = 0, j;

     do {
        buff[i++] = (value % 10) | 0x30;
        value  /= 10;
     }while(value);

     i--;

     for(j = 0 ;j <= i; j++)
         buffer[j] = buff[i - j];
     buffer[j] = 0;
     return buffer;
 }
int main(void) {
    uint32_t ui32ADC0Value[4],i;char buffer[8];
    volatile uint32_t ui32TempAvg;
    volatile uint32_t ui32TempValueC;
    volatile uint32_t ui32TempValueF;
 SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
SYSCTL_XTAL_16MHZ);
 SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
 SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
 GPIOPinConfigure(GPIO_PA0_U0RX);
 GPIOPinConfigure(GPIO_PA1_U0TX);
 GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
 SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF); //enable GPIO port for LED
 GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_2); //enable pin for LED PF2
 UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200,
 (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));
 SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1);
 TimerConfigure(TIMER1_BASE, TIMER_CFG_PERIODIC);

 ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
 ROM_ADCHardwareOversampleConfigure(ADC0_BASE, 32);

 ROM_ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);
 ROM_ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_TS);
 ROM_ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_TS);
 ROM_ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_TS);
 ROM_ADCSequenceStepConfigure(ADC0_BASE,1,3,ADC_CTL_TS|ADC_CTL_IE|ADC_CTL_END);
 ROM_ADCSequenceEnable(ADC0_BASE, 1);

 uint32_t ui32Period = ((SysCtlClockGet())*50) /100;
 TimerLoadSet(TIMER1_BASE, TIMER_A, ui32Period -1);

 IntEnable(INT_TIMER1A);
 TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);


 TimerEnable(TIMER1_BASE, TIMER_A);

 IntEnable(INT_UART0); //enable the UART interrupt
 UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT); //only enable RX and TX
interrupts
 IntMasterEnable(); //enable processor interrupts
 while (1) //let interrupt handler do the UART echo function
 {
     while(!ROM_ADCIntStatus(ADC0_BASE, 1, false))
           {
           }

           ROM_ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value);
           ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] +
ui32ADC0Value[3] + 2)/4;
           ui32TempValueC = (1475 - ((2475 * ui32TempAvg)) / 4096)/10;
           ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5;
```

```
            UARTCharPut(UART0_BASE, 'T');
            UARTCharPut(UART0_BASE, 'e');
            UARTCharPut(UART0_BASE, 'm');
            UARTCharPut(UART0_BASE, 'p');
            UARTCharPut(UART0_BASE, ':');
             itoa( ui32TempValueC, buffer);

             for(i = 0; i < 2; i++)
             {
                 UARTCharPut(UART0_BASE, buffer[i]);
             }
             UARTCharPut(UART0_BASE, '\n');UARTCharPut(UART0_BASE, '\r');
        }
 }

void Timer1IntHandler(void)
{
    // Clear the timer interrupt
    TimerIntClear(TIMER1_BASE, TIMER_TIMA_TIMEOUT);

    // Read the current state of the GPIO pin and
    // write back the opposite state
    ROM_ADCIntClear(ADC0_BASE, 1);
    ROM_ADCProcessorTrigger(ADC0_BASE, 1);
}
```



```
Temp:17
Temp:17
Temp:17
Temp:17
Temp:17
Temp:17
Temp:17
Temp:17
Temp:17
Temp:17
Temp:
```

Task02

Modified code:

```
#include <stdint.h>
#include <stdbool.h>
#include "driverlib/gpio.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "driverlib/pin_map.h"
```

```
#include "driverlib/adc.h"
#include "inc/hw_memmap.h"
#include "inc/hw_ints.h"
#include "inc/hw_types.h"

/**
 * main.c
 */

void UARTPrintf(const char *str)
{
    while(*str)
        UARTCharPut(UART0_BASE ,*str++);
}

char* atoi(int value, char *buffer)
{
    char buff[10];
    int i = 0, j;
    do {
        buff[i++] = value % 10;
        value /= 10;
    }while(value);

    for(j = 0; j < i; j++)
        buffer[j] = buff[i-j];
    buffer[j] = 0;
    return buffer;
}

int main(void)
{
    int32_t cmd, buffer[16];
    uint32_t ui32ADC0Value[4];
    volatile uint32_t ui32TempAvg;
    volatile uint32_t ui32TempValueC;
    volatile uint32_t ui32TempValueF;

    SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
SYSCTL_XTAL_16MHZ);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    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);
    ADCSequenceEnable(ADC0_BASE, 1);

    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0|GPIO_PIN_1);
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);
    UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200,
    (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));

    while(true)
    {
        UARTPrintf("Enter the Cmd : ");
        while(!UARTCharsAvail(UART0_BASE));
        {
            cmd = UARTCharGet(UART0_BASE);
            switch(cmd)
            {
            case 'R' : { GPIOPinWrite(SYSCTL_PERIPH_GPIOF, GPIO_PIN_1, GPIO_PIN_1);
break; }
            case 'G' : { GPIOPinWrite(SYSCTL_PERIPH_GPIOF, GPIO_PIN_2, GPIO_PIN_2);
break; }
            case 'B' : { GPIOPinWrite(SYSCTL_PERIPH_GPIOF, GPIO_PIN_3, GPIO_PIN_3);
break; }
            case 'r' : { GPIOPinWrite(SYSCTL_PERIPH_GPIOF, GPIO_PIN_1, 0); break; }
            case 'g' : { GPIOPinWrite(SYSCTL_PERIPH_GPIOF, GPIO_PIN_2, 0); break; }
            case 'b' : { GPIOPinWrite(SYSCTL_PERIPH_GPIOF, GPIO_PIN_3, 0); break; }
            case 'T' : { ADCIntClear(ADC0_BASE, 1);
                         ADCProcessorTrigger(ADC0_BASE, 1);
                         while(!ADCIntStatus(ADC0_BASE, 1, false));
                         ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value);
                         ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] +
ui32ADC0Value[2] + ui32ADC0Value[3] + 2)/4;
                         ui32TempValueC = (1475 - ((2475 * ui32TempAvg)) / 4096)/10;
                         ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5;
                         atoi(ui32TempValueC, buffer);
                         UARTPrintf(buffer);
                         break;
                       }
            }
        }
    }

    return 0;
}
```

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