Itzel Becerril
Github root directory: https://github.com/HadidBuilds/TivaC_project_labs

Date Submitted: **10/28/19**

Task 00: Execute provided code

Youtube Link:
https://www.youtube.com/watch?v=wQ8y6fV7J-8

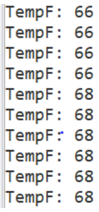
```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "inc/hw_ints.h"
#include "driverlib/interrupt.h"
//need to enable processor interrupts
//we will select receiver interrupts and receiver timeout interrupts

int main(void)
{
    //set up the system clock
    SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
SYSCTL_XTAL_16MHZ);

    //enable UART0 and GPIOA peripherals
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    //configure pins PA0 as reciever and PA1 as the transmitter using GPIOPinConfig
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    //initialize the GPIO peripheral and pin for the LED
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_2);

    //initialize the parameters for the UART: 115200, 8-1-N
    UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200, (UART_CONFIG_WLEN_8 |
UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));

    IntMasterEnable();
    IntEnable(INT_UART0);
    UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT);

    //calls to create the prompt
    UARTCharPut(UART0_BASE, 'E');
    UARTCharPut(UART0_BASE, 'n');
    UARTCharPut(UART0_BASE, 't');
    UARTCharPut(UART0_BASE, 'e');
    UARTCharPut(UART0_BASE, 'r');
    UARTCharPut(UART0_BASE, ' ');
    UARTCharPut(UART0_BASE, 'T');
    UARTCharPut(UART0_BASE, 'e');
    UARTCharPut(UART0_BASE, 'x');
    UARTCharPut(UART0_BASE, 't');
    UARTCharPut(UART0_BASE, ':');
    UARTCharPut(UART0_BASE, ' ');

    //if there is a character in the receiver it is read and then written to the
transmitter
    //this echos what you type in the terminal window
    while(1)
    {
       //if(UARTCharsAvail(UART0_BASE)) UARTCharPut(UART0_BASE,
UARTCharGet(UART0_BASE));
 }
}

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 ms
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0);  //turn off LED
    }
}

-----------------------------------------------------------------------------------
```

Task 01:

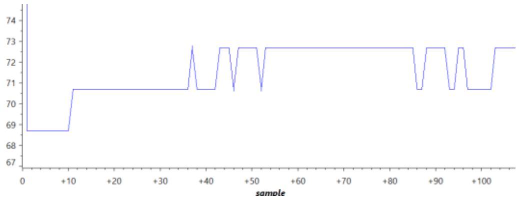
In this task, I am to display the temperature on the terminal using a 0.5 timer interrupt.

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

Modified Schematic (if applicable):



```
TempF: 66
TempF: 66
TempF: 66
TempF: 66
TempF: 68
TempF: 68
TempF: 68
TempF: 68
TempF: 68
TempF: 68
```



Pics are different values bc done at different times

Modified Code:

```
#include <stdint.h>
#include <stdbool.h>
```

```
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/uart.h"
#include "inc/tm4c123gh6pm.h"
#include "driverlib/timer.h"
#include "driverlib/adc.h"
#include "driverlib/debug.h"
#include "driverlib/interrupt.h"

void UART_OutUDec(uint32_t);
void UART_OutChar(char data);

uint32_t ui32ADC0Value[1];

//volatile so that each variable cannot be optimized out by the compiler
volatile uint32_t ui32TempAvg;
volatile uint32_t ui32TempValueC;
volatile uint32_t ui32TempValueF;

int main(void)
{
    //set up the system clock
    SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
SYSCTL_XTAL_16MHZ);

    //enable UART0 and GPIOA peripherals
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    //configure pins PA0 as reciever and PA1 as the transmitter using GPIOPinConfig
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    //initialize the GPIO peripheral and pin for the LEDS
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3);

    //initialize the parameters for the UART: 115200, 8-1-N
    UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200,
           (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));

    //Enable ADC0 peripheral
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
    //hardware average of 32
    ADCHardwareOversampleConfigure(ADC0_BASE, 32);

    //ADC will run at default rate of 1Msps
    //Configure ADC sequencer 3
    //want the processor to trigger the sequence and use highest priority
    ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_PROCESSOR, 0);
    ADCSequenceStepConfigure(ADC0_BASE, 3, 0, ADC_CTL_TS | ADC_CTL_IE | ADC_CTL_END);
    //timer1 value
    int32_t ui32Period = (SysCtlClockGet() / 1);

    //Timer 1 enabling and config
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1);
    TimerConfigure(TIMER1_BASE, TIMER_CFG_PERIODIC);
    TimerLoadSet(TIMER1_BASE, TIMER_A, 5 * (SysCtlClockGet() / 10));

    //Enabling interrupts
    IntEnable(INT_TIMER1A);
    TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
    TimerEnable(TIMER1_BASE, TIMER_A);
    IntMasterEnable();

    //Enabling ADC interrupts
    ADCSequenceEnable(ADC0_BASE, 3);
    ADCIntEnable(ADC0_BASE, 3);
    while (1)
    {

    }
}

void Timer1IntHandler(void)
{
    int32_t ui32PeriodHigh = 0.5 * (SysCtlClockGet());

    //Clear timer interrupt
    TimerIntClear(TIMER1_BASE, TIMER_TIMA_TIMEOUT);

    //Set the value of timer
    TimerLoadSet(TIMER1_BASE, TIMER_A, ui32PeriodHigh);

    //clear interrupt flags
    ADCIntClear(ADC0_BASE, 3);
    //trigger ADC conversion with software
    ADCProcessorTrigger(ADC0_BASE, 3);

    //wait for conversion
    while (!ADCIntStatus(ADC0_BASE, 3, false))
    {
    }

    //get data from a buffer in memory
    ADCSequenceDataGet(ADC0_BASE, 3, ui32ADC0Value);

    //Gets the value form array
    ADCSequenceDataGet(ADC0_BASE, 3, ui32ADC0Value);

    //Calculates temp
    ui32TempValueC = (1475 - ((2475 * ui32ADC0Value[0])) / 4096) / 10;
    ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5;
    //printing to terminal
    UARTCharPut(UART0_BASE, 'T');
    UARTCharPut(UART0_BASE, 'e');
    UARTCharPut(UART0_BASE, 'm');
    UARTCharPut(UART0_BASE, 'p');
    UARTCharPut(UART0_BASE, 'F');
    UARTCharPut(UART0_BASE, ':');
    UARTCharPut(UART0_BASE, ' ');
    UART_OutUDec(ui32TempValueF);
    UARTCharPut(UART0_BASE, '\n');
    UARTCharPut(UART0_BASE, '\r');
}

void UART_OutUDec(uint32_t n)
{
    if (n >= 10) {
        UART_OutUDec(n / 10);
        n = n % 10;
    }
    UART_OutChar(n + '0');
}

void UART_OutChar(char data)
{
    while ((UART0_FR_R&UART_FR_TXFF) != 0);
    UART0_DR_R = data;
}
```

```
------------------------------------------------------------------------------------
```

Task 02:

In this task, I am to create a user interface using UART. If ‘B’ is pressed, then the Blue led will turn on. If ‘b’ is pressed, then the Blue led will turn off. If ‘R’ is pressed, then the red led will turn on and if ‘r’ is pressed then it will turn off. If ‘G’ is pressed, then the Green LED will turn on else ‘g’ will turn it off. If ‘T’ is pressed, then it will display the temperature.

Youtube Link:
https://www.youtube.com/watch?v=9ljQR9L-Rtk

Modified Schematic (if applicable):
None

Modified Code:

```
// Insert code here
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/uart.h"
```

```
#include "inc/tm4c123gh6pm.h"
#include "driverlib/adc.h"
#include "driverlib/debug.h"
#include "driverlib/interrupt.h"

void UART_OutUDec(uint32_t);
void UART_OutChar(char data);

uint32_t ui32ADC0Value[1];

//volatile so that each variable cannot be optimized out by the compiler
volatile uint32_t ui32TempAvg;
volatile uint32_t ui32TempValueC;
volatile uint32_t ui32TempValueF;

int main(void)
{
    //set up the system clock
    SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
SYSCTL_XTAL_16MHZ);

    //enable UART0 and GPIOA peripherals
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    //configure pins PA0 as reciever and PA1 as the transmitter using GPIOPinConfig
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    //initialize the GPIO peripheral and pin for the LEDS
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3);

    //Enable ADC0 peripheral
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
    //hardware average of 32
    ADCHardwareOversampleConfigure(ADC0_BASE, 32);


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

    //ADC will run at default rate of 1Msps
    //Configure ADC sequencer 3
    //want the processor to trigger the sequence and use highest priority
    UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200,
        (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));

    //Enabling UART interrupts
    IntMasterEnable();
    IntEnable(INT_UART0);
    UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT);

    //Enabling ADC interrupts
    ADCSequenceEnable(ADC0_BASE, 3);
    ADCIntEnable(ADC0_BASE, 3);

    while (1)
    {
    }
}

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

    //Turn on Blue LED
    if(UARTCharGet(UART0_BASE) == 'B')
    {
        UARTCharPut(UART0_BASE, 'B');
        UARTCharPut(UART0_BASE, 'l');
        UARTCharPut(UART0_BASE, 'u');
        UARTCharPut(UART0_BASE, 'e');
        UARTCharPut(UART0_BASE, ' ');
        UARTCharPut(UART0_BASE, 'O');
        UARTCharPut(UART0_BASE, 'n');
        UARTCharPut(UART0_BASE, '\n');
        UARTCharPut(UART0_BASE, '\r');

        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, GPIO_PIN_2); //blink LED
        SysCtlDelay(10000000);
    }
    //Turn off Blue LED
    if(UARTCharGet(UART0_BASE) == 'b')
    {
        UARTCharPut(UART0_BASE, 'B');
        UARTCharPut(UART0_BASE, 'l');
        UARTCharPut(UART0_BASE, 'u');
        UARTCharPut(UART0_BASE, 'e');
        UARTCharPut(UART0_BASE, ' ');
        UARTCharPut(UART0_BASE, 'O');
        UARTCharPut(UART0_BASE, 'f');
        UARTCharPut(UART0_BASE, 'f');
        UARTCharPut(UART0_BASE, '\n');
        UARTCharPut(UART0_BASE, '\r');

        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0); //blink LED
        SysCtlDelay(10000000);
    }
    //Turn on Red LED
    if(UARTCharGet(UART0_BASE) == 'R')
    {
        UARTCharPut(UART0_BASE, 'R');
        UARTCharPut(UART0_BASE, 'e');
        UARTCharPut(UART0_BASE, 'd');
        UARTCharPut(UART0_BASE, ' ');
        UARTCharPut(UART0_BASE, 'O');
```

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

```
        UARTCharPut(UART0_BASE, 'n');
        UARTCharPut(UART0_BASE, '\n');
        UARTCharPut(UART0_BASE, '\r');

        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 2); //blink LED
        SysCtlDelay(10000000);
    }
    //Turn off Red LED
    if(UARTCharGet(UART0_BASE) == 'r')
    {
        UARTCharPut(UART0_BASE, 'R');
        UARTCharPut(UART0_BASE, 'e');
        UARTCharPut(UART0_BASE, 'd');
        UARTCharPut(UART0_BASE, ' ');
        UARTCharPut(UART0_BASE, 'O');
        UARTCharPut(UART0_BASE, 'f');
        UARTCharPut(UART0_BASE, 'f');
        UARTCharPut(UART0_BASE, '\n');
        UARTCharPut(UART0_BASE, '\r');

        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0); //blink LED
        SysCtlDelay(10000000);
    }
    //Turn on Green LED
    if(UARTCharGet(UART0_BASE) == 'G')
    {
        UARTCharPut(UART0_BASE, 'G');
        UARTCharPut(UART0_BASE, 'r');
        UARTCharPut(UART0_BASE, 'e');
        UARTCharPut(UART0_BASE, 'e');
        UARTCharPut(UART0_BASE, 'n');
        UARTCharPut(UART0_BASE, ' ');
        UARTCharPut(UART0_BASE, 'O');
        UARTCharPut(UART0_BASE, 'n');
        UARTCharPut(UART0_BASE, '\n');
        UARTCharPut(UART0_BASE, '\r');

        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 8); //blink LED
        SysCtlDelay(10000000);
    }
    //Turn off Green LED
    if(UARTCharGet(UART0_BASE) == 'g')
    {
        UARTCharPut(UART0_BASE, 'G');
        UARTCharPut(UART0_BASE, 'r');
        UARTCharPut(UART0_BASE, 'e');
        UARTCharPut(UART0_BASE, 'e');
        UARTCharPut(UART0_BASE, 'n');
        UARTCharPut(UART0_BASE, ' ');
        UARTCharPut(UART0_BASE, 'O');
        UARTCharPut(UART0_BASE, 'f');
        UARTCharPut(UART0_BASE, 'f');
        UARTCharPut(UART0_BASE, '\n');
        UARTCharPut(UART0_BASE, '\r');
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0); //blink LED
        SysCtlDelay(10000000);
    }
    //Display Temp if T is pressed
    if(UARTCharGet(UART0_BASE) == 'T')
    {
        //clear interrupt flags
        ADCIntClear(ADC0_BASE, 3);
        //trigger ADC conversion with software
        ADCProcessorTrigger(ADC0_BASE, 3);

        //wait for conversion
        while (!ADCIntStatus(ADC0_BASE, 3, false))
        {
        }

        //get data from a buffer in memory
        ADCSequenceDataGet(ADC0_BASE, 3, ui32ADC0Value);

        //Calculates temp
        ui32TempValueC = (1475 - ((2475 * ui32ADC0Value[0])) / 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, 'F');
        UARTCharPut(UART0_BASE, ':');
        UARTCharPut(UART0_BASE, ' ');
        UART_OutUDec(ui32TempValueF);
        UARTCharPut(UART0_BASE, '\n');
        UARTCharPut(UART0_BASE, '\r');
    }
}

void UART_OutUDec(uint32_t n)
{
    if (n >= 10) {
        UART_OutUDec(n / 10);
        n = n % 10;
    }
    UART_OutChar(n + '0');
}

void UART_OutChar(char data)
{
    while ((UART0_FR_R&UART_FR_TXFF) != 0);
    UART0_DR_R = data;
}

-----------------------------------------------------------------------------------
```

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