Ivan Soto
Github root directory: https://github.com/sotoi2/Class3.0.4

Date Submitted:

Task 00: Execute provided code

Youtube Link: https://www.youtube.com/watch?v=HNvJPx6U2UY

```
#include <stdint.h>
#include <stdbool.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"
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
 }
}
int main(void) {
 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));
 IntMasterEnable(); //enable processor interrupts
 IntEnable(INT_UART0); //enable the UART interrupt
 UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT); //only enable RX and TX
interrupts
 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');
```

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

```
 UARTCharPut(UART0_BASE, 'e');
 UARTCharPut(UART0_BASE, 'x');
 UARTCharPut(UART0_BASE, 't');
 UARTCharPut(UART0_BASE, ':');
 UARTCharPut(UART0_BASE, ' ');
 while (1) //let interrupt handler do the UART echo function
 {
// if (UARTCharsAvail(UART0_BASE)) UARTCharPut(UART0_BASE, UARTCharGet(UART0_BASE));
 }
}
```

Task 01:

Youtube Link: https://www.youtube.com/watch?v=J5eNfYyx-sw

Modified Schematic (if applicable):

Modified Code:

```
    #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 "driverlib/interrupt.h"
    #include "driverlib/adc.h"
    #include "driverlib/rom.h"
    #include "inc/hw_ints.h"
    #include "driverlib/timer.h"


volatile uint32_t ui32Period;
    volatile uint32_t ui32TempAvg;
    volatile uint32_t ui32TempValueC;
    volatile uint32_t ui32TempValueF;

    volatile  int i;


    void Timer1IntHandler(void)
{

        uint8_t characters[10];
uint32_t ui32ADC0[4];
```

```
    TimerIntClear(TIMER1_BASE,TIMER_A);// Always clear the interrupt for the values
that may depend on it in the future




 // Clear the ADC interrup status flag
              ROM_ADCIntClear(ADC0_BASE, 2);
             // Trigger ADC conversion with software
             ROM_ADCProcessorTrigger(ADC0_BASE, 2);


            // waith for the conversion to complete
            while(!ROM_ADCIntStatus(ADC0_BASE, 2, false))
            {
            }



            // we can read the ADC value from the ADC sample sequencer 1 FIFO
            ROM_ADCSequenceDataGet(ADC0_BASE, 2, ui32ADC0);

            // calculate the average of the temperature sensor data
            ui32TempAvg = (ui32ADC0[0] + ui32ADC0[1] + ui32ADC0[2] + ui32ADC0[3] +
2)/4;
            // calculate celsius value
            ui32TempValueC = (1475 - ((2475 * ui32TempAvg)) / 4096)/10;
            // calculate farenheit value
            ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5;

int count = 5;
         i = 0;

         while(ui32TempValueF != 0)
         {
         characters[i++] = (ui32TempValueF%10)+ '0';
         ui32TempValueF /=10;
         }

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


}
```

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

Ivan Soto
Github root directory: https://github.com/sotoi2/Class3.0.4

```
int main(void) {


     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);

     UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200,
     (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));









     ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
     ROM_ADCHardwareOversampleConfigure(ADC0_BASE, 64);

     ROM_ADCSequenceConfigure(ADC0_BASE, 2, ADC_TRIGGER_PROCESSOR, 0);
     ROM_ADCSequenceStepConfigure(ADC0_BASE, 2, 0, ADC_CTL_TS);
     ROM_ADCSequenceStepConfigure(ADC0_BASE, 2, 1, ADC_CTL_TS);
     ROM_ADCSequenceStepConfigure(ADC0_BASE, 2, 2, ADC_CTL_TS);

     ROM_ADCSequenceStepConfigure(ADC0_BASE,2,3,ADC_CTL_TS|ADC_CTL_IE|ADC_CTL_END);



     ROM_ADCSequenceEnable(ADC0_BASE, 2);




ADCIntEnable(ADC0_BASE,2);
 //     UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT);
ROM_SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);                // enable GPIO
peripherals
       GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);   //
configure pins as outputs for LEDs


       SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1);   // enable clock to timer1
```

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

```
          TimerConfigure(TIMER1_BASE, TIMER_CFG_PERIODIC);     // configure timer 1
in periodic mode


              ui32Period = (SysCtlClockGet() / 1) / 2;     // sets the delay
              TimerLoadSet(TIMER1_BASE, TIMER_A, ui32Period -1);   // load into
Timer's Interval Load register

              IntEnable(INT_TIMER1A);  // enables specific vector associated with
Timer 0A
              TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT); // enables a specific
event within the timer to generate an interrupt (on timeouts)
             // IntMasterEnable();  // master interrupt enable for all interrupts

              TimerEnable(TIMER1_BASE, TIMER_A);// finally enable the timer



     while (1)
     {

     }



}
```

Task 02:

Youtube Link: https://www.youtube.com/watch?v=KHmfLHLXuz8

Modified Schematic (if applicable):

Modified Code:

```
    #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 "driverlib/interrupt.h"
    #include "driverlib/adc.h"
    #include "driverlib/rom.h"
    #include "inc/hw_ints.h"
```

```
//#define TARGET_IS_BLIZZARD_RB1
    //variables used in previous labs for temp
    volatile uint32_t ui32TempAvg;
    volatile uint32_t ui32TempValueC;
    volatile uint32_t ui32TempValueF;

    volatile  int i;


    void UARTIntHandler(void)
    {


        uint32_t ui32ADC0[4];
        uint32_t ui32Status;


        uint8_t characters[10];

        ui32Status = UARTIntStatus(UART0_BASE, true);

        char usercharacter;

        UARTIntClear(UART0_BASE, ui32Status);

        while(UARTCharsAvail(UART0_BASE)) //
        {
        usercharacter = UARTCharGet(UART0_BASE);
        UARTCharPut(UART0_BASE, usercharacter);



        if(usercharacter == 'R')
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 2);


        if(usercharacter == 'G')
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 8);


        if(usercharacter == 'B')
           GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, GPIO_PIN_2);


        if(usercharacter == 'r')
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1,0);

        if(usercharacter == 'g')
           GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0);

        if(usercharacter == 'b')
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0);
```

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

```
if(usercharacter == 'T')
 {
 ROM_ADCIntClear(ADC0_BASE, 1);
 ROM_ADCProcessorTrigger(ADC0_BASE, 1);

 while(!ROM_ADCIntStatus(ADC0_BASE, 1, false))
 {
 }

 ROM_ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0);
 ui32TempAvg = (ui32ADC0[0] + ui32ADC0[1] + ui32ADC0[2] + ui32ADC0[3] + 2)/4;
 ui32TempValueC = (1475 - ((2475 * ui32TempAvg)) / 4096)/10;
 ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5;


 int count = 5;
 i = 0;

 while(ui32TempValueF != 0)
 {
 characters[i++] = (ui32TempValueF%10)+ '0';
 ui32TempValueF /=10;
 }

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

 }

 }
}
```

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

Ivan Soto
Github root directory: https://github.com/sotoi2/Class3.0.4

```
int main(void) {


 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);
 UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200,
 (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));
 SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
 GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_2| GPIO_PIN_1| GPIO_PIN_3);


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


 UARTCharPut(UART0_BASE, 'R');
 UARTCharPut(UART0_BASE, ':');
 UARTCharPut(UART0_BASE, ' ');
 UARTCharPut(UART0_BASE, 'R');
 UARTCharPut(UART0_BASE, 'e');
 UARTCharPut(UART0_BASE, 'd');
 UARTCharPut(UART0_BASE, ' ');
 UARTCharPut(UART0_BASE, 'L');
 UARTCharPut(UART0_BASE, 'E');
 UARTCharPut(UART0_BASE, 'D');
 UARTCharPut(UART0_BASE, ',');
 UARTCharPut(UART0_BASE, ' ');
 UARTCharPut(UART0_BASE, 'G');
 UARTCharPut(UART0_BASE, ':');
 UARTCharPut(UART0_BASE, ' ');
 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, 'L');
 UARTCharPut(UART0_BASE, 'E');
 UARTCharPut(UART0_BASE, 'D');
 UARTCharPut(UART0_BASE, ',');
 UARTCharPut(UART0_BASE, 'B');
 UARTCharPut(UART0_BASE, ':');
 UARTCharPut(UART0_BASE, ' ');
 UARTCharPut(UART0_BASE, 'B');
```

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

```
    UARTCharPut(UART0_BASE, 'l');
    UARTCharPut(UART0_BASE, 'u');
    UARTCharPut(UART0_BASE, 'e');
    UARTCharPut(UART0_BASE, ' ');
    UARTCharPut(UART0_BASE, 'L');
    UARTCharPut(UART0_BASE, 'E');
    UARTCharPut(UART0_BASE, 'D');
    UARTCharPut(UART0_BASE, ',');
    UARTCharPut(UART0_BASE, ' ');
    UARTCharPut(UART0_BASE, 'T');
    UARTCharPut(UART0_BASE, ':');
    UARTCharPut(UART0_BASE, ' ');
    UARTCharPut(UART0_BASE, 'T');
    UARTCharPut(UART0_BASE, 'e');
    UARTCharPut(UART0_BASE, 'm');
    UARTCharPut(UART0_BASE, 'p');
    UARTCharPut(UART0_BASE, 'e');
    UARTCharPut(UART0_BASE, 'r');
    UARTCharPut(UART0_BASE, 'a');
    UARTCharPut(UART0_BASE, 't');
    UARTCharPut(UART0_BASE, 'u');
    UARTCharPut(UART0_BASE, 'r');
    UARTCharPut(UART0_BASE, 'e');


    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);

    while (1)
    {

    }

   }
```

```
// Insert code here

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

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

Ivan Soto
Github root directory: https://github.com/sotoi2/Class3.0.4

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