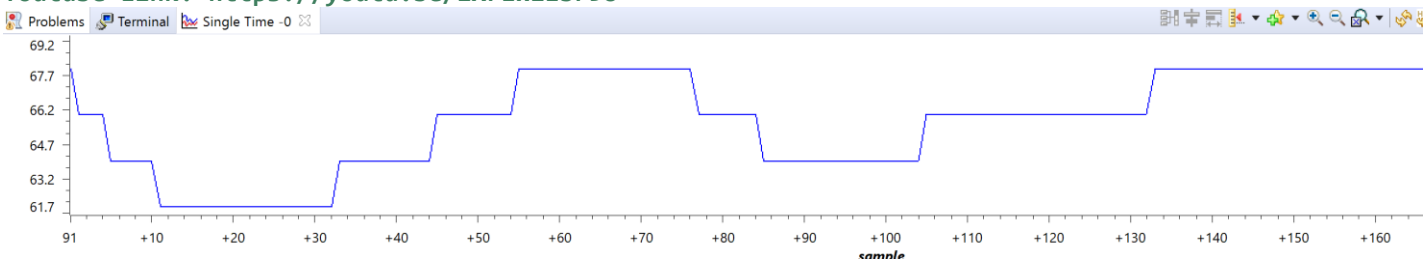
Tenniel Takenaka-Fuller
Github root directory: https://github.com/TennielTakenaka/CLASS2/tree/master/lab7

Date Submitted: 10/13/2019

Task 00: Execute provided code

Youtube Link: https://youtu.be/iXFIWlzbr9U



This is the same graph from the YouTube video. The temperatures are being sampled every 0.5 seconds. Watch the full YouTube video to see a zoom in on the temperature values. The cup was cold which brought the temperature down. I then rubbed the device to create heat in the form of friction to bring the temperature back up.

Task 01:

Youtube Link:
Lab7-Task01ccsGUI: https://youtu.be/l3fbZAdFhe8

Lab7-Task01putty: https://youtu.be/bBbYXPPHRp4

TEMPERATURES ARE SAMPLED AT 0.5 SECONDS USING THE TIMER 1 INTERRUPT. I FORGOT TO MENTION THIS IN THE VIDEOS BUT IT IS EVIDENT BY THE GRAPH ABOVE AND THE CODE I CREATED.

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"inc/hw_ints.h"
#include"driverlib/interrupt.h" //needed for timer 1 interrupt
#include"driverlib/adc.h"
#include "inc/tm4c123gh6pm.h"
#include"driverlib/interrupt.h" //needed for interrupt functions
#include "driverlib/timer.h" //needed for timer functions
#ifdef DEBUG
void__error__(char*pcFilename, uint32_t ui32Line)
{
}
#endif
```

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

```
uint32_t ui32ADC0Value[1]; //value of 1 for adc0 step 3
volatile uint32_t ui32TempAvg;
volatile uint32_t ui32TempValueC;
volatile uint32_t ui32TempValueF;

//Function: 8 bit ascii character to be transferred
void UartIntToChar (char x)
{
    while((UART0_FR_R&UART_FR_TXFF) != 0);
    UART0_DR_R = x;
}
//Function:Use recursion to convert a 32 bit integer to a string
void UartTransmit (uint32_t x)
{
    if (x>=10)
    {
        UartTransmit(x / 10);
        x = x % 10;
    }
    UartIntToChar(x+'0');
}

int main(void) {

      SysCtlClockSet(SYSCTL_SYSDIV_5 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
SYSCTL_XTAL_16MHZ);

    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
    ADCHardwareOversampleConfigure(ADC0_BASE, 32);

    //Uart Configure
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

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

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_2);

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

    //Timer 1 Configure
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1);
    TimerConfigure(TIMER1_BASE, TIMER_CFG_PERIODIC);
    TimerLoadSet(TIMER1_BASE, TIMER_A, (SysCtlClockGet() * .5));
    IntEnable(INT_TIMER1A);
    TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
    IntMasterEnable();
```

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

```
    TimerEnable(TIMER1_BASE,TIMER_A);

    ADCSequenceEnable(ADC0_BASE, 3);
    ADCIntEnable(ADC0_BASE, 3); //enable interrupt for sequence 3

    while (1)
    {
    }
}

void Timer1IntHandler(void) //created timer1 interrupt handler
{
    ADCIntClear(ADC0_BASE, 3); //clear adc conversion done flag before writing code
that depends on it. change to sequence 2
    TimerIntClear(TIMER1_BASE, TIMER_TIMA_TIMEOUT); //clear timer

    TimerLoadSet(TIMER1_BASE, TIMER_A, (SysCtlClockGet() * .5)); //timer loaded to .5
seconds
    ADCProcessorTrigger(ADC0_BASE, 3); //changed to sequence 3

    while(!ADCIntStatus(ADC0_BASE, 3, false)) //wait for conversion to finish
    {
    } //if loop exited conversion is complete

    ADCSequenceDataGet(ADC0_BASE, 3, ui32ADC0Value); //gets samples from the array
    ui32TempValueC = (1475 -((2475 * ui32ADC0Value[0])) / 4096)/10; //only one adcval
    ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5;
    UartTransmit(ui32TempValueF); //send the temperature to be converted to a string
    UARTCharPut(UART0_BASE, '\n'); //print line feeds
    UARTCharPut(UART0_BASE, '\r');
}
```

Task 02:

Youtube Link: https://youtu.be/L8viAwgOLTc

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"inc/hw_ints.h"
#include"driverlib/interrupt.h" //needed for interrupts
#include"driverlib/adc.h"
#include "inc/tm4c123gh6pm.h"
#include"driverlib/interrupt.h" //needed for interrupt functions

#ifdef DEBUG
void__error__(char*pcFilename, uint32_t ui32Line)
```

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

```
{
}
#endif

uint32_t ui32ADC0Value[1]; //value of 1 for adc0 step 3
volatile uint32_t ui32TempAvg;
volatile uint32_t ui32TempValueC;
volatile uint32_t ui32TempValueF;

//Function: 8 bit ascii character to be transferred
void UartIntToChar (char x)
{
    while((UART0_FR_R&UART_FR_TXFF) != 0);
    UART0_DR_R = x;
}
//Function:Use recursion to convert a 32 bit integer to a string
void UartTransmit (uint32_t x)
{
    if (x>=10)
    {
        UartTransmit(x / 10);
        x = x % 10;
    }
    UartIntToChar(x+'0');
}

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
    {
        switch(UARTCharGet(UART0_BASE)) //read the UART character that has been
retreived
        {
        case 'R': //turn on red led
            UARTCharPut(UART0_BASE, 'R');
            UARTCharPut(UART0_BASE, '\n');
            UARTCharPut(UART0_BASE, '\r');
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, GPIO_PIN_1); //blink LED
            SysCtlDelay(SysCtlClockGet() / (1000*3)); //key debouncing
            break;
        case 'r': //turn off red led
            UARTCharPut(UART0_BASE, 'r');
            UARTCharPut(UART0_BASE, '\n');
            UARTCharPut(UART0_BASE, '\r');
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0); //turn off LED
            SysCtlDelay(SysCtlClockGet() / (1000*3)); //key debouncing
            break;
        case 'B': //turn on blue led
            UARTCharPut(UART0_BASE, 'B');
            UARTCharPut(UART0_BASE, '\n');
            UARTCharPut(UART0_BASE, '\r');
```

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

```
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, GPIO_PIN_2); //blink LED
            SysCtlDelay(SysCtlClockGet() / (1000*3)); //key debouncing
            break;
        case 'b': //turn off blue led
            UARTCharPut(UART0_BASE, 'b');
            UARTCharPut(UART0_BASE, '\n');
            UARTCharPut(UART0_BASE, '\r');
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0); //turn off LED
            SysCtlDelay(SysCtlClockGet() / (1000*3)); //key debouncing
            break;
        case 'G': //turn on green led
            UARTCharPut(UART0_BASE, 'G');
            UARTCharPut(UART0_BASE, '\n');
            UARTCharPut(UART0_BASE, '\r');
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, GPIO_PIN_3); //blink LED
            SysCtlDelay(SysCtlClockGet() / (1000*3)); //key debouncing
            break;
        case 'g': //turn off green led
            UARTCharPut(UART0_BASE, 'g');
            UARTCharPut(UART0_BASE, '\n');
            UARTCharPut(UART0_BASE, '\r');
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0); //turn off LED
            SysCtlDelay(SysCtlClockGet() / (1000*3)); //key debouncing
            break;
        case 'T': //calc and display temp
            UARTCharPut(UART0_BASE, 'T');
            UARTCharPut(UART0_BASE, 'e');
            UARTCharPut(UART0_BASE, 'm');
            UARTCharPut(UART0_BASE, 'p');
            UARTCharPut(UART0_BASE, ':');
            UARTCharPut(UART0_BASE, ' ');
            UartTransmit (ui32TempValueF); //send temp to the transmit function to
display in terminal
            UARTCharPut(UART0_BASE, '\n');
            UARTCharPut(UART0_BASE, '\r');
            SysCtlDelay(SysCtlClockGet() / (1000*3)); //key debouncing
            break;

        }
    }
}

int main(void) {

      SysCtlClockSet(SYSCTL_SYSDIV_5 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
SYSCTL_XTAL_16MHZ);

    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
    ADCHardwareOversampleConfigure(ADC0_BASE, 32);

    //Uart Configure
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
```

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

```
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

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

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF); //enable GPIO port for LED
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);
//enable pin for LED PF2

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


    ADCSequenceEnable(ADC0_BASE, 3);
    ADCIntEnable(ADC0_BASE, 3); //enable interrupt for sequence 3
    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

    while (1)
    {
        ADCIntClear(ADC0_BASE, 3); //clear adc conversion done flag before writing
code that depends on it
        ADCProcessorTrigger(ADC0_BASE, 3); //changed to sequence 3
        while(!ADCIntStatus(ADC0_BASE, 3, false)) //wait for conversion to finish
        {
        } //if loop exited conversion is complete

        ADCSequenceDataGet(ADC0_BASE, 3, ui32ADC0Value); //gets samples from the
array
        ui32TempValueC = (1475 -((2475 * ui32ADC0Value[0])) / 4096)/10; //only one
adcval
        ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5; //continuously calculate
the temperature
    }
}
```

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