John Duriman

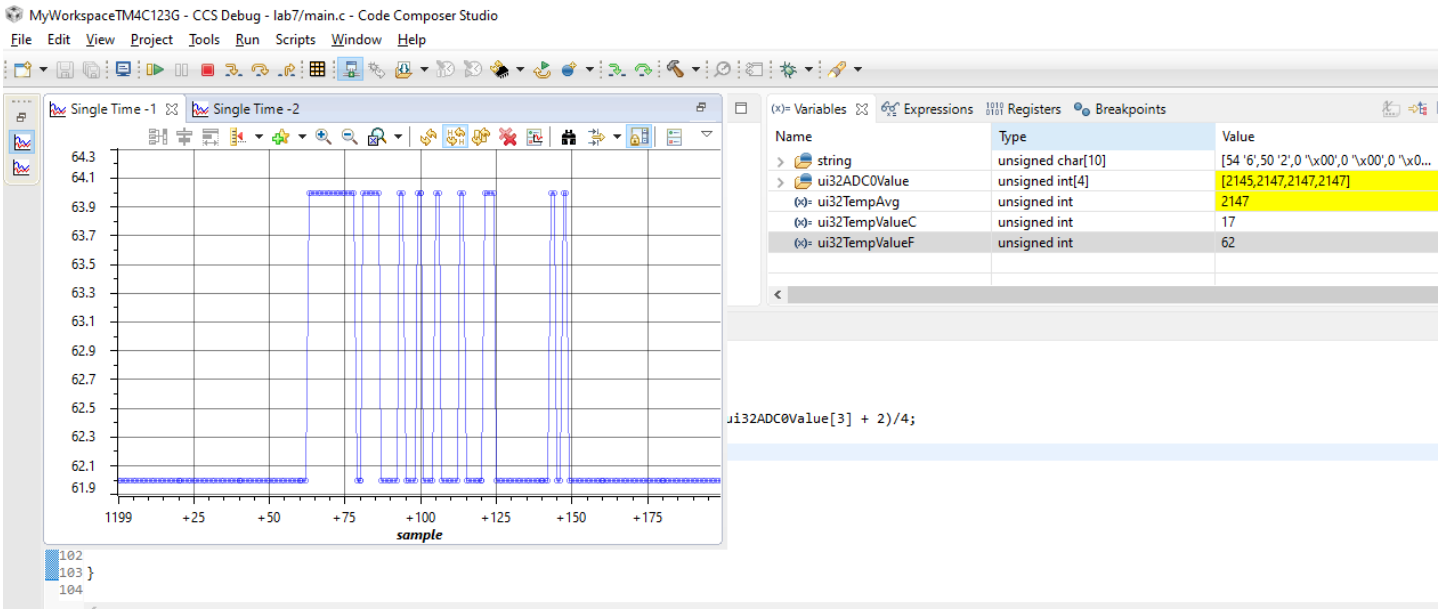
CPE 403

Lab 7

5002373995

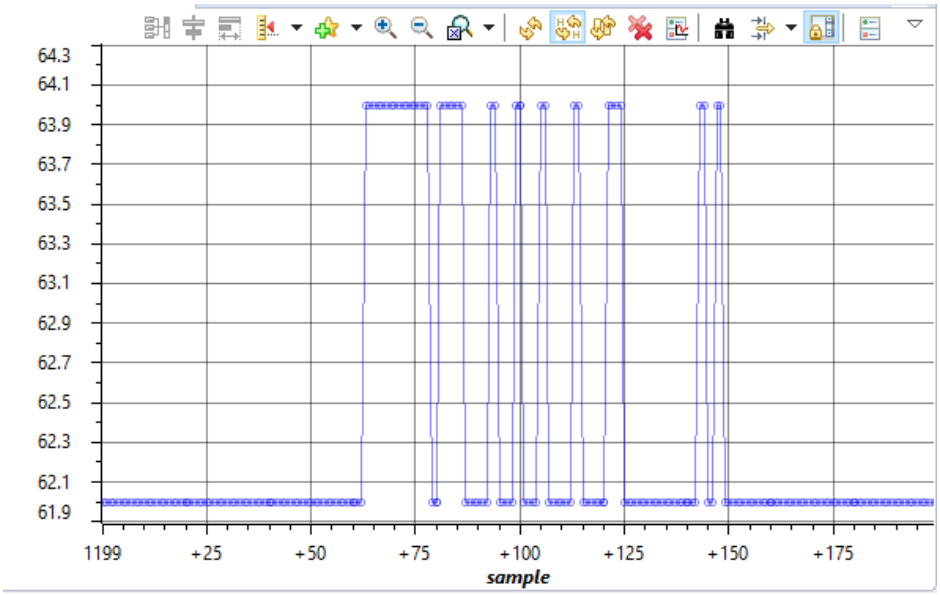
Task 00: Execute the provided code, display the temperatures in the built-in Graph Tool

Youtube Link: https://youtu.be/n2tKQkZZTzM



Task 01: Continuously display the temperature of the device (internal temperature sensor) on the a) hyperterminal, and b) GUI Composer (Temp Sensor) using a timer interrupt every 0.5 secs.

Youtube Link: https://youtu.be/n2tKQkZZTzM



Modified Code:

I made the variables global, so both main and Timer1IntHandler can access the values.

main.c | main1.txt | tm4c123gh6pm_startup_ccs.c

```
#include <stdint.h>
#include <stdbool.h>
#include <stdlib.h>
#include <stdio.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/rom_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "driverlib/adc.h"
#include "driverlib/sysctl.h"
#include "driverlib/timer.h"

void printm(char *str);
void itoa(int n, char s[]);

uint32_t ui32ADC0Value[4];              //Array for storing ADC FIFO data
volatile uint32_t ui32TempAvg;          //Temp sensor data
volatile uint32_t ui32TempValueC;       //Celsius
volatile uint32_t ui32TempValueF;       //Fahrenheit
char string[10];

int main(void)
{
    //Setup clock and ADC
    SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN | SYSCTL_XTAL_16MHZ);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);         //Enable ADC0
    ADCHardwareOversampleConfigure(ADC0_BASE, 64);

    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);

    //Enable GPIO and configure pins as outputs
    //SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    //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));
```

```
    //Configure ADC sequencer, sample sequencer 1, trigger the sequence at highest priority
    ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);

    //Configure all four steps in the ADC sequencer to sequencer 2
    ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_TS);
    //Final sequencer step
    ADCSequenceStepConfigure(ADC0_BASE, 1 ,3 , ADC_CTL_TS|ADC_CTL_IE|ADC_CTL_END);

    //Enable ADC Sequencer 1
    ADCSequenceEnable(ADC0_BASE, 1);

    //Setup Timer 1
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1);
    TimerConfigure(TIMER1_BASE, TIMER_CFG_PERIODIC);
    TimerLoadSet(TIMER1_BASE, TIMER_A, (SysCtlClockGet()/2)-1);
    TimerEnable(TIMER1_BASE, TIMER_A);


    //Enable interrupts
    IntMasterEnable();
    IntEnable(INT_UART0);           //Enable UART interrupt
    IntEnable(INT_TIMER1A);
    TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
    UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT);          //Enable RX and TX interrupts
    while (1)
    {
        /*
        uint32_t ui32ADC0Value[4];              //Array for storing ADC FIFO data
        volatile uint32_t ui32TempAvg;          //Temp sensor data
        volatile uint32_t ui32TempValueC;       //Celsius
        //volatile uint32_t ui32TempValueF;       //Fahrenheit
        char string[10];*/

        //if (UARTCharsAvail(UART0_BASE)) UARTCharPut(UART0_BASE, UARTCharGet(UART0_BASE));

        //Clear ADC interrupt status flag
        ADCIntClear(ADC0_BASE, 1);
        //Trigger ADC conversion
        ADCProcessorTrigger(ADC0_BASE, 1);

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

        //Read ADC value
        ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value);

        //Calculate average of the temperature sensor data
        ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] + ui32ADC0Value[3] + 2)/4;
        ui32TempValueC = (1475 - ((2475 * ui32TempAvg)) / 4096)/10;
        ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5;

        itoa(ui32TempValueF, string);
    }
}

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

    printm(string);
    printm("\r\n");
}

void printm(char *str)
{
    while(*str != '\0')
    {
        UARTCharPut(UART0_BASE, *str);
        ++str;
    }
}

void reverse(char s[])
{
    int i, j;
    char c;

    for(i = 0, j = strlen(s)-1; i < j; i++, j--)
    {
        c = s[i];
        s[i] = s[j];
        s[j] = c;
    }
}

void itoa(int n, char s[])
{
    int i, sign;

    if((sign = n ) < 0)
        n = -n;

    i = 0;
    do
    {
        s[i++] = n % 10 + '0';
    }
    while ((n/=10) > 0);
    if (sign < 0)
        s[i++] = '-';
    s[i] = '\0';
    reverse(s);
}
```

Task 02: **Interaction/User Interface: Develop a user interface using UART to perform the following:**

Enter the cmd: R: RED LED, G: Green LED, B: Blue LED, T: Temperature:

Based on the command (cmd) the program should turn ON Red LED when R is entered in the terminal, etc. Command of 'r' will turn off the Red LED.

Youtube Link: https://youtu.be/n2tKQkZZTzM

Modified Code:

Added two variables that hold an array of characters for each temperature to put for the function itoa()

```c
#include <stdint.h>
#include <stdbool.h>
#include <stdlib.h>
#include <stdio.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/rom_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "driverlib/adc.h"
#include "driverlib/sysctl.h"
#include "driverlib/timer.h"

uint32_t ui32ADC0Value[4];              //Array for storing ADC FIFO data
volatile uint32_t ui32TempAvg;          //Temp sensor data
volatile uint32_t ui32TempValueC;       //Celsius
volatile uint32_t ui32TempValueF;       //Fahrenheit
volatile uint32_t ui8PinData;           //LED
char string[10];                        //Holds Temperature in F
char string1[10];                       //Holds Temperature in C

void printm(char *str)
{
    while(*str != '\0')
    {
        UARTCharPut(UART0_BASE, *str);
        ++str;
    }
}

void reverse(char s[])
{
    int i, j;
    char c;

    for(i = 0, j = strlen(s)-1; i < j; i++, j--)
    {
        c = s[i];
        s[i] = s[j];
        s[j] = c;
    }
}
```

Variable command is a character that is taken from the UART which is then compared to values in the switch case. ‘R’ will turn on the RED LED light and then print an extra line as well as the prompt for the next command etc.

```
49 void itoa(int n, char s[])
50 {
51     int i, sign;
52
53     if((sign = n ) < 0)
54         n = -n;
55
56     i = 0;
57     do
58     {
59         s[i++] = n % 10 + '0';
60     }
61     while ((n/=10) > 0);
62     if (sign < 0)
63         s[i++] = '-';
64     s[i] = '\0';
65     reverse(s);
66 }
67
68 void UARTIntHandler(void)
69 {
70     uint32_t ui32Status;
71     char command;
72
73     ui32Status = UARTIntStatus(UART0_BASE, true); //get interrupt status
74
75     UARTIntClear(UART0_BASE, ui32Status); //clear the asserted interrupts
76
77     while(UARTCharsAvail(UART0_BASE)) //loop while there are chars
78     {
79         //UARTCharPutNonBlocking(UART0_BASE, UARTCharGetNonBlocking(UART0_BASE)); //echo character
80         command = UARTCharGet(UART0_BASE);
81         UARTCharPut(UART0_BASE, command);
82         switch(command)
83         {
84         case 'R':
85             ui8PinData = 2;
86             GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3, 0);
87             GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3, ui8PinData);
88             printm("\r\n");
89             printm("Enter Text: ");
90             break;
```

Command ‘T’ just prints out both temperature values pulled from the ADC

```
91         case 'G':
92             ui8PinData = 8;
93             GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3, 0);
94             GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3, ui8PinData);
95             printm("\r\n");
96             printm("Enter Text: ");
97             break;
98         case 'B':
99             ui8PinData = 4;
100            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3, 0);
101            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3, ui8PinData);
102            printm("\r\n");
103            printm("Enter Text: ");
104            break;
105        case 'T':
106            printm("\r\n");
107            printm("Temp = ");
108            printm(string);
109            printm("F = ");
110            printm(string1);
111            printm("C");
112            printm("\r\n");
113            printm("Enter Text: ");
114            break;
115        default:
116            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3, 0);
117            printm("\r\n");
118            printm("Enter Text: ");
119            break;
120        }
121    }
122 }
123
124
125 int main(void) {
126
127     //Setup clock and ADC
128     SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN | SYSCTL_XTAL_16MHZ);
129     SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);        //Enable ADC0
130     ADCHardwareOversampleConfigure(ADC0_BASE, 64);
131
132     SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN | SYSCTL_XTAL_16MHZ);
133
134     SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
135     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
```

Added GPIO output settings for LED lights

```
    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_1|GPIO_PIN_2|GPIO_PIN_3); //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

    //Configure ADC sequencer, sample sequencer 1, trigger the sequence at highest priority
    ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);

    //Configure all four steps in the ADC sequencer to sequencer 2
    ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_TS);
    //Final sequencer step
    ADCSequenceStepConfigure(ADC0_BASE, 1 ,3 , ADC_CTL_TS|ADC_CTL_IE|ADC_CTL_END);

    //Enable ADC Sequencer 1
    ADCSequenceEnable(ADC0_BASE, 1);

    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, ' ');
```

ADC logic is done in the while loop to keep the temperature consistent and because I was having trouble getting itoa() to work outside of int main(). Itoa is run twice: one for Fahrenheit and the other for Celsius

```
    while (1) //let interrupt handler do the UART echo function
    {
        //Clear ADC interrupt status flag
        ADCIntClear(ADC0_BASE, 1);
        //Trigger ADC conversion
        ADCProcessorTrigger(ADC0_BASE, 1);

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

        //Read ADC value
        ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value);

        //Calculate average of the temperature sensor data
        ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] + ui32ADC0Value[3] + 2)/4;
        ui32TempValueC = (1475 - ((2475 * ui32TempAvg)) / 4096)/10;
        ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5;
        itoa(ui32TempValueF, string);
        itoa(ui32TempValueC, string1);
    }
}
```