Abenezer Namaga
CPE 403 – 1001
TIVAC LAB 07

Github root directory: https://github.com/Ayertena/TIVAC

Date Due: October 12th, 2018

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

Task 01: 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://www.youtube.com/watch?v=F1C8iKeYZpY

```
#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/timer.h" //using timer1
#include "driverlib/interrupt.h" //using interrupt
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"

#include "driverlib/adc.h" //ADC for temperature

int main(void)
{
    //50MHz clock
    SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN
| SYSCTL_XTAL_16MHZ);

    //ADC0 Peripherals
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
    ADCHardwareOversampleConfigure(ADC0_BASE, 64);

    //Configure and Enable Timer1
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1);
```

```
    TimerConfigure(TIMER1_BASE, TIMER_CFG_PERIODIC);
    TimerLoadSet(TIMER1_BASE, TIMER_A, (SysCtlClockGet()/2)-1); //0.5
delay
    IntEnable(INT_TIMER1A);
    TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);

    //ADC Sequence, four step ADC sequencer
    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);

    //Enables UART0 and GPIOA
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    //Transmit/Receive pins
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    //UART Configure, baud rate of 115200, 8 transmit bits, 1 stop
bit, 0 parity bits
    UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200,
                        (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
UART_CONFIG_PAR_NONE));

    IntMasterEnable();
    ADCSequenceEnable(ADC0_BASE, 1);    //Enable ADC0
    TimerEnable(TIMER1_BASE, TIMER_A); //Enable Timer1

    //INITALIZATION TEST
    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, ' ');

    while (1) //let interrupt handler do the UART echo function
    {

    }
}

void Timer1IntHandler(void)
{

    //ADC0 Array
    uint32_t ui32ADC0Value[4];

    //Variables for Temperature: Average, Celsius, and Fahrenheit
    volatile uint32_t ui32TempAvg;
    volatile uint32_t ui32TempValueC;
    volatile uint32_t ui32TempValueF;

    //string to place temperature
    char temp[2];

    //Clear ADC Interrupt Flag and trigger ADC conversion
    ADCIntClear(ADC0_BASE, 1);
    ADCProcessorTrigger(ADC0_BASE, 1);

    //Wait for conversion to complete
    while(!ADCIntStatus(ADC0_BASE, 1, false))
    {
    }
    //Copies data from the specified sample sequencer output FIFO to a
buffer in memory
    ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value);

    //Calculate Average, use it to get Celsuis Temp, then convert to
Fahrenheit
    ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] +
ui32ADC0Value[2] + ui32ADC0Value[3] + 2)/4;
    ui32TempValueC = (1475 - ((2475 * ui32TempAvg)) / 4096)/10;
    ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5;

    //converts uint32_t to ascii, then prints value
```

```
    temp[0] = ui32TempValueF/10 + 0x30; // 0x30 is 0 in hex so it can
convert it from int to char
    temp[1] = ui32TempValueF%10 + 0x30; // attain the 2nd figure and
convert it to char
    UARTCharPut(UART0_BASE, temp[0]);
    UARTCharPut(UART0_BASE, temp[1]);
    UARTCharPut(UART0_BASE, '\r');
    UARTCharPut(UART0_BASE, '\n');
    SysCtlDelay(5000000);
}
```

Task 02: Interaction/User Interface
Developed a user interface using UART to perform the following by entering command keys on keyboard.

- R: Red LED, G: Green LED, B: Blue LED, T: Temperature:
-

Youtube Link: https://www.youtube.com/watch?v=iGfYyjGJ52c

```
#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/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"

#include "driverlib/adc.h" //ADC for temperature

int main(void)
{
    //50MHz clock
    SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN
| SYSCTL_XTAL_16MHZ);

    //ADC0 Peripherals
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
    ADCHardwareOversampleConfigure(ADC0_BASE, 64);

    //ADC Sequence, four step ADC sequencer
    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);

    //Enables UART0 and GPIOA
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
```

```
    //Transmit/Receive pins
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    //Enable Port F
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,
GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);

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

    //UART Configure, baud rate of 115200, 8 transmit bits, 1 stop
bit, 0 parity bits
    UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200,
                        (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
UART_CONFIG_PAR_NONE));

    ADCSequenceEnable(ADC0_BASE, 1); //enable ADC0
    IntEnable(INT_UART0); //enable the UART interrupt
    UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT); //only
enable RX and TX interrupts

    //PROMPT
    UARTCharPut(UART0_BASE, 'R');
    UARTCharPut(UART0_BASE, ',');
    UARTCharPut(UART0_BASE, ' ');
    UARTCharPut(UART0_BASE, 'B');
    UARTCharPut(UART0_BASE, ',');
    UARTCharPut(UART0_BASE, ' ');
    UARTCharPut(UART0_BASE, 'G');
    UARTCharPut(UART0_BASE, ',');
    UARTCharPut(UART0_BASE, ' ');
    UARTCharPut(UART0_BASE, 'o');
    UARTCharPut(UART0_BASE, 'r');
    UARTCharPut(UART0_BASE, ' ');
    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));
        char n = UARTCharGet(UART0_BASE);
        if(n == 0x52) //Pressed 'R': TURN ON RED LED
        {
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 2);
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0);
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0);
            UARTCharPut(UART0_BASE, n);
        }
        else if(n == 0x42) //Pressed 'B': TURN ON BLUE LED
        {
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0);
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 4);
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0);
            UARTCharPut(UART0_BASE, n);
        }
        else if(n == 0x47) //Pressed 'G': TURN ON GREEN LED
        {
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0);
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0);
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 8);
            UARTCharPut(UART0_BASE, n);
        }
        else if(n == 0x72) //Pressed 'r': TURN OFF RED LED
        {
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0);
            UARTCharPut(UART0_BASE, n);
        }
        else if(n == 0x62) //Pressed 'b': TURN OFF BLUE LED
        {
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0);
            UARTCharPut(UART0_BASE, n);
        }
        else if(n == 0x67) //Pressed 'g': TURN OFF GREEN LED
        {
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0);
            UARTCharPut(UART0_BASE, n);
        }
        else if(n == 0x54 || n == 0x74) //Pressed 'T' or 't':
TEMPERATURE
        {
            //ADC0 Array
```

```
            uint32_t ui32ADC0Value[4];

            //Variables for Temperature: Average, Celsius, and
Fahrenheit
            volatile uint32_t ui32TempAvg;
            volatile uint32_t ui32TempValueC;
            volatile uint32_t ui32TempValueF;
            volatile uint32_t holder;

            //string to place temperature
            char temp[2];

            //Clear ADC Interrupt Flag and trigger ADC conversion
            ADCIntClear(ADC0_BASE, 1);
            ADCProcessorTrigger(ADC0_BASE, 1);

            //Wait for conversion to complete
            while(!ADCIntStatus(ADC0_BASE, 1, false))
            {
            }

            //Copies data from the specified sample sequencer output
FIFO to a buffer in memory
            ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value);

            //Calculate Average, use it to get Celsuis Temp, then
convert to Fahrenheit
            ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] +
ui32ADC0Value[2] + ui32ADC0Value[3] + 2)/4;
            ui32TempValueC = (1475 - ((2475 * ui32TempAvg)) /
4096)/10;
            ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5;

            //converts uint32_t to ascii, then prints value
            temp[0] = ui32TempValueF/10 + 0x30; // 0x30 is 0 in hex so
it can convert it from int to char
            temp[1] = ui32TempValueF%10 + 0x30; // attain the 2nd
figure and convert it to char
            UARTCharPut(UART0_BASE, 'T');
            UARTCharPut(UART0_BASE, 'E');
            UARTCharPut(UART0_BASE, 'M');
            UARTCharPut(UART0_BASE, 'P');
            UARTCharPut(UART0_BASE, ':');
            UARTCharPut(UART0_BASE, ' ');
```

```
            UARTCharPut(UART0_BASE, temp[0]);
            UARTCharPut(UART0_BASE, temp[1]);
            UARTCharPut(UART0_BASE, 'F');
        }

        //PROMPT
        UARTCharPut(UART0_BASE, '\r');
        UARTCharPut(UART0_BASE, '\n');
        UARTCharPut(UART0_BASE, 'R');
        UARTCharPut(UART0_BASE, ',');
        UARTCharPut(UART0_BASE, ' ');
        UARTCharPut(UART0_BASE, 'B');
        UARTCharPut(UART0_BASE, ',');
        UARTCharPut(UART0_BASE, ' ');
        UARTCharPut(UART0_BASE, 'G');
        UARTCharPut(UART0_BASE, ',');
        UARTCharPut(UART0_BASE, ' ');
        UARTCharPut(UART0_BASE, 'o');
        UARTCharPut(UART0_BASE, 'r');
        UARTCharPut(UART0_BASE, ' ');
        UARTCharPut(UART0_BASE, 'T');
        UARTCharPut(UART0_BASE, '?');
        UARTCharPut(UART0_BASE, ' ');
    }

}
```