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, ' ');

}

}