**Date Submitted: 10/12/2018**

**Task 00: Execute provided code**

**Youtube Link:**

**No Submission required for Task 00**

**------------------------------------------------------------------------------------**

**Task 01:**

Youtube Link: <https://youtu.be/YUxMpZKrJGM>

**Modified Schematic (if applicable): N/A**

**Modified Code:**

**// Insert code here**

**#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"

**#include** "driverlib/adc.h"

**#include** <string.h>

**#include** <math.h>

**#include** <stdlib.h>

**#include** "driverlib/rom.h"

**#include** "inc/tm4c123gh6pm.h" //added libraries for task 1

**#include** "driverlib/timer.h"

**#include** "driverlib/debug.h"

**volatile** uint32\_t ui32TempAvg;

**volatile** uint32\_t ui32TempValueC;

**volatile** uint32\_t ui32TempValueF;

**volatile** uint32\_t tempF;

**volatile** **int** i; //for loop variable

**int** **main**(**void**) {

uint32\_t ui32Period;

**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);

**GPIOPinTypeGPIOOutput**(GPIO\_PORTF\_BASE, GPIO\_PIN\_2| GPIO\_PIN\_1| GPIO\_PIN\_3);

**UARTConfigSetExpClk**(UART0\_BASE, **SysCtlClockGet**(), 115200,

(UART\_CONFIG\_WLEN\_8 | UART\_CONFIG\_STOP\_ONE | UART\_CONFIG\_PAR\_NONE));

**UARTCharPut**(UART0\_BASE, 'R');

**UARTCharPut**(UART0\_BASE, 'e');

**UARTCharPut**(UART0\_BASE, 'a');

**UARTCharPut**(UART0\_BASE, 'd');

**UARTCharPut**(UART0\_BASE, 'i');

**UARTCharPut**(UART0\_BASE, 'n');

**UARTCharPut**(UART0\_BASE, 'g');

**UARTCharPut**(UART0\_BASE, ' ');

**UARTCharPut**(UART0\_BASE, 'T');

**UARTCharPut**(UART0\_BASE, 'e');

**UARTCharPut**(UART0\_BASE, 'm');

**UARTCharPut**(UART0\_BASE, 'p');

**UARTCharPut**(UART0\_BASE, ':');

**UARTCharPut**(UART0\_BASE, '\n'); //print new line

**UARTCharPut**(UART0\_BASE, '\r'); //return on new line

//The section below initializes the ADC for temperature reading

//------------------------------------------------------

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_ADC0);

ROM\_ADCHardwareOversampleConfigure(ADC0\_BASE, 32);

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);

//-------------------------------------------------------

//-------------------------------------------------------

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_TIMER1); //enable peripheral timer1

**TimerConfigure**(TIMER1\_BASE, TIMER\_CFG\_PERIODIC); //configure timer1 to periodic mode

ui32Period = **SysCtlClockGet**()/2;

**TimerLoadSet**(TIMER1\_BASE, TIMER\_A, ui32Period -1);

**IntEnable**(INT\_TIMER1A);//enables the specific vector associated with Timer0A

**TimerIntEnable**(TIMER1\_BASE, TIMER\_TIMA\_TIMEOUT);//enables a specific event within the timer to generate an interrupt

**IntMasterEnable**();//master interrupt enable API for all interrupts

**TimerEnable**(TIMER1\_BASE, TIMER\_A);//enable the timer

//-------------------------------------------------------

**while** (1)

{

}

}

**void** **Timer1IntHandler**(**void**)

{

uint32\_t ui32ADC0Value[4];

uint8\_t arr[10];

// Clear the timer interrupt

**TimerIntClear**(TIMER1\_BASE, TIMER\_TIMA\_TIMEOUT);

// Read the current state of the GPIO pin and

// write back the opposite state

**ADCIntClear**(ADC0\_BASE, 2); //clear the ADC interrupt status flag

**ADCProcessorTrigger**(ADC0\_BASE, 2);

**while**(!**ADCIntStatus**(ADC0\_BASE, 2, false))

{

}

**ADCSequenceDataGet**(ADC0\_BASE, 2, ui32ADC0Value);

ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] + ui32ADC0Value[3] + 2)/4;

ui32TempValueC = (1475 - ((2475 \* ui32TempAvg)) / 4096)/10;

ui32TempValueF = ((ui32TempValueC \* 9) + 160) / 5;

tempF = ui32TempValueF;

**int** sizeOfInteger = 5;

i = 0;

**while**(ui32TempValueF != 0) //this loop converts an integer to an array of characters

{

arr[i++] = (ui32TempValueF%10)+ '0';

ui32TempValueF /=10;

}

**for**( i = 0; i<sizeOfInteger; i++) //this loop prints the temperature character by character

{

**UARTCharPut**(UART0\_BASE, arr[i]);

}

**UARTCharPut**(UART0\_BASE, 'F'); //print F for fahrenheit

**UARTCharPut**(UART0\_BASE, '\n'); //print new line

**UARTCharPut**(UART0\_BASE, '\r'); //return on new line

}

**------------------------------------------------------------------------------------**

**Task 02:**

Youtube Link:

**Modified Schematic (if applicable): N/A**

**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"

**#include** "driverlib/adc.h"

**#include** <string.h>

**#include** <math.h>

**#include** <stdlib.h>

**#include** "driverlib/rom.h"

**volatile** uint32\_t ui32TempAvg;

**volatile** uint32\_t ui32TempValueC;

**volatile** uint32\_t ui32TempValueF;

**volatile** **int** i; //for loop variable

**void** **UARTIntHandler**(**void**)

{

uint8\_t arr[10];

uint32\_t ui32ADC0Value[4];

uint32\_t ui32Status;

ui32Status = **UARTIntStatus**(UART0\_BASE, true); //get interrupt status

**char** newchar; //variable to hold character typed by the user

**UARTIntClear**(UART0\_BASE, ui32Status);

**while**(**UARTCharsAvail**(UART0\_BASE)) //while there are characters in the terminal

{

newchar = **UARTCharGet**(UART0\_BASE); //set newchar equal to the character typed by the user

**UARTCharPut**(UART0\_BASE, newchar); //echo the character typed by the user

**if**(newchar == 'B')

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_2, GPIO\_PIN\_2); //turn Blue LED on

**if**(newchar == 'b')

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_2, 0x00); //turn Blue LED off

**if**(newchar == 'R')

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1, 0x02); //turn red LED on

**if**(newchar == 'r')

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1, 0x00); //turn red LED off

**if**(newchar == 'G')

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_3, 0x08); //turn Green LED on

**if**(newchar == 'g')

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_3, 0x00); //turn Green LED off

**if**(newchar == 'T')

{

ROM\_ADCIntClear(ADC0\_BASE, 1);

ROM\_ADCProcessorTrigger(ADC0\_BASE, 1);

**while**(!ROM\_ADCIntStatus(ADC0\_BASE, 1, false))

{

}

ROM\_ADCSequenceDataGet(ADC0\_BASE, 1, ui32ADC0Value);

ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] + ui32ADC0Value[3] + 2)/4;

ui32TempValueC = (1475 - ((2475 \* ui32TempAvg)) / 4096)/10;

ui32TempValueF = ((ui32TempValueC \* 9) + 160) / 5;

**int** sizeOfInteger = 5;

i = 0;

**while**(ui32TempValueF != 0) //this loop converts an integer to an array of characters

{

arr[i++] = (ui32TempValueF%10)+ '0';

ui32TempValueF /=10;

}

**for**( i = 0; i<sizeOfInteger; i++) //this loop prints the temperature character by character

{

**UARTCharPut**(UART0\_BASE, arr[i]);

}

**UARTCharPut**(UART0\_BASE, 'F'); //print F for fahrenheit

**UARTCharPut**(UART0\_BASE, '\n'); //print new line

**UARTCharPut**(UART0\_BASE, '\r'); //return on new line

}

}

}

**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);

**GPIOPinTypeGPIOOutput**(GPIO\_PORTF\_BASE, GPIO\_PIN\_2| GPIO\_PIN\_1| GPIO\_PIN\_3);

**UARTConfigSetExpClk**(UART0\_BASE, **SysCtlClockGet**(), 115200,

(UART\_CONFIG\_WLEN\_8 | UART\_CONFIG\_STOP\_ONE | UART\_CONFIG\_PAR\_NONE));

**IntMasterEnable**();

**IntEnable**(INT\_UART0);

**UARTIntEnable**(UART0\_BASE, UART\_INT\_RX | UART\_INT\_RT);

**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, 'h');

**UARTCharPut**(UART0\_BASE, 'e');

**UARTCharPut**(UART0\_BASE, ' ');

**UARTCharPut**(UART0\_BASE, 'c');

**UARTCharPut**(UART0\_BASE, 'm');

**UARTCharPut**(UART0\_BASE, 'd');

**UARTCharPut**(UART0\_BASE, ':');

**UARTCharPut**(UART0\_BASE, ' ');

//The section below initializes the ADC for temperature reading

//------------------------------------------------------

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)

{

}

}

**------------------------------------------------------------------------------------**