**Date Submitted:**

**Task 00: Execute provided code**

**Youtube Link:** <https://www.youtube.com/watch?v=HNvJPx6U2UY>

**#include** <stdint.h>

**#include** <stdbool.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/sysctl.h"

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

**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

{

**UARTCharPutNonBlocking**(UART0\_BASE, **UARTCharGetNonBlocking**(UART0\_BASE)); //echo character

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_2, GPIO\_PIN\_2); //blink LED

**SysCtlDelay**(**SysCtlClockGet**() / (1000 \* 3)); //delay ~1 msec

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

}

}

**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); //enable GPIO port for LED

**GPIOPinTypeGPIOOutput**(GPIO\_PORTF\_BASE, GPIO\_PIN\_2); //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

**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

{

// if (UARTCharsAvail(UART0\_BASE)) UARTCharPut(UART0\_BASE, UARTCharGet(UART0\_BASE));

}

}

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

**Task 01:**

Youtube Link: <https://www.youtube.com/watch?v=J5eNfYyx-sw>

**Modified Schematic (if applicable):**

**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** "driverlib/interrupt.h"

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

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

**#include** "inc/hw\_ints.h"

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

**volatile** uint32\_t ui32Period;

**volatile** uint32\_t ui32TempAvg;

**volatile** uint32\_t ui32TempValueC;

**volatile** uint32\_t ui32TempValueF;

**volatile** **int** i;

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

{

uint8\_t characters[10];

uint32\_t ui32ADC0[4];

**TimerIntClear**(TIMER1\_BASE,TIMER\_A);// Always clear the interrupt for the values that may depend on it in the future

// Clear the ADC interrup status flag

ROM\_ADCIntClear(ADC0\_BASE, 2);

// Trigger ADC conversion with software

ROM\_ADCProcessorTrigger(ADC0\_BASE, 2);

// waith for the conversion to complete

**while**(!ROM\_ADCIntStatus(ADC0\_BASE, 2, false))

{

}

// we can read the ADC value from the ADC sample sequencer 1 FIFO

ROM\_ADCSequenceDataGet(ADC0\_BASE, 2, ui32ADC0);

// calculate the average of the temperature sensor data

ui32TempAvg = (ui32ADC0[0] + ui32ADC0[1] + ui32ADC0[2] + ui32ADC0[3] + 2)/4;

// calculate celsius value

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

// calculate farenheit value

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

**int** count = 5;

i = 0;

**while**(ui32TempValueF != 0)

{

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

ui32TempValueF /=10;

}

**for**( i = 0; i<count; i++)

{

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

}

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

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

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

}

**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);

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

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

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_ADC0);

ROM\_ADCHardwareOversampleConfigure(ADC0\_BASE, 64);

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);

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

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

ROM\_SysCtlClockSet(SYSCTL\_SYSDIV\_5|SYSCTL\_USE\_PLL|SYSCTL\_OSC\_MAIN|SYSCTL\_XTAL\_16MHZ);

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOF); // enable GPIO peripherals

**GPIOPinTypeGPIOOutput**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1|GPIO\_PIN\_2|GPIO\_PIN\_3); // configure pins as outputs for LEDs

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

**TimerConfigure**(TIMER1\_BASE, TIMER\_CFG\_PERIODIC); // configure timer 1 in periodic mode

ui32Period = (**SysCtlClockGet**() / 1) / 2; // sets the delay

**TimerLoadSet**(TIMER1\_BASE, TIMER\_A, ui32Period -1); // load into Timer's Interval Load register

**IntEnable**(INT\_TIMER1A); // enables specific vector associated with Timer 0A

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

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

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

**while** (1)

{

}

}

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

**Task 02:**

Youtube Link: <https://www.youtube.com/watch?v=KHmfLHLXuz8>

**Modified Schematic (if applicable):**

**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** "driverlib/interrupt.h"

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

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

**#include** "inc/hw\_ints.h"

//#define TARGET\_IS\_BLIZZARD\_RB1

//variables used in previous labs for temp

**volatile** uint32\_t ui32TempAvg;

**volatile** uint32\_t ui32TempValueC;

**volatile** uint32\_t ui32TempValueF;

**volatile** **int** i;

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

{

uint32\_t ui32ADC0[4];

uint32\_t ui32Status;

uint8\_t characters[10];

ui32Status = **UARTIntStatus**(UART0\_BASE, true);

**char** usercharacter;

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

**while**(**UARTCharsAvail**(UART0\_BASE)) //

{

usercharacter = **UARTCharGet**(UART0\_BASE);

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

**if**(usercharacter == 'R')

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1, 2);

**if**(usercharacter == 'G')

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_3, 8);

**if**(usercharacter == 'B')

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_2, GPIO\_PIN\_2);

**if**(usercharacter == 'r')

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1,0);

**if**(usercharacter == 'g')

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_3, 0);

**if**(usercharacter == 'b')

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_2, 0);

**if**(usercharacter == 'T')

{

ROM\_ADCIntClear(ADC0\_BASE, 1);

ROM\_ADCProcessorTrigger(ADC0\_BASE, 1);

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

{

}

ROM\_ADCSequenceDataGet(ADC0\_BASE, 1, ui32ADC0);

ui32TempAvg = (ui32ADC0[0] + ui32ADC0[1] + ui32ADC0[2] + ui32ADC0[3] + 2)/4;

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

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

**int** count = 5;

i = 0;

**while**(ui32TempValueF != 0)

{

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

ui32TempValueF /=10;

}

**for**( i = 0; i<count; i++)

{

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

}

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

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

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

}

}

}

**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);

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

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

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOF);

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

**IntMasterEnable**();

**IntEnable**(INT\_UART0);

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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)

{

}

}

**// Insert code here**

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