**10/11/2018**

**Task 00: Execute provided code**

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

Modifications highlighted in Yellow

**Task 01:**

Youtube Link: https://youtu.be/pF2q4B8xnYY

**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/timer.h"

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

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

// Prototype

**void** **itoa**(**long** **unsigned** **int** value, **char**\* result, **int** base);

**#ifdef** DEBUG

**void\_\_error\_\_**(**char** \*pcFilename, uint32\_t ui32Line)

{

}

**#endif**

// Globals

uint32\_t ui32Period;

**char** buffer[4];

uint32\_t ui32ADC0Value[4];

**volatile** uint32\_t ui32TempAvg;

**volatile** uint32\_t ui32TempValueC;

**volatile** uint32\_t ui32TempValueF;

// Timer 1 ISR

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

{

uint32\_t i;

// Clear the timer interrupt

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

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

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

**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;

itoa(ui32TempValueF, &buffer, 10); // This function was taken for the C standard library.

**for** (i=0; i<4; i++)

{

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

}

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

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

}

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

// Configure Clock

**SysCtlClockSet**(SYSCTL\_SYSDIV\_4 | SYSCTL\_USE\_PLL | SYSCTL\_OSC\_MAIN | SYSCTL\_XTAL\_16MHZ);

// Configure peripherals

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_UART0);

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

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_TIMER1); // Enabling Timer 1

// Configure ADC

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_ADC0);

**ADCHardwareOversampleConfigure**(ADC0\_BASE, 32);

**ADCSequenceConfigure**(ADC0\_BASE, 2, ADC\_TRIGGER\_PROCESSOR, 0); // Changed to sequencer #2

**ADCSequenceStepConfigure**(ADC0\_BASE, 2, 0, ADC\_CTL\_TS);

**ADCSequenceStepConfigure**(ADC0\_BASE, 2, 1, ADC\_CTL\_TS);

**ADCSequenceStepConfigure**(ADC0\_BASE, 2, 2, ADC\_CTL\_TS);

**ADCSequenceStepConfigure**(ADC0\_BASE, 2, 3, ADC\_CTL\_TS|ADC\_CTL\_IE|ADC\_CTL\_END);

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

////////////////////

// Configure Timer 1 module

**TimerConfigure**(TIMER1\_BASE, TIMER\_CFG\_PERIODIC);

ui32Period = **SysCtlClockGet**()/2; // Period of 0.5s 2Hz

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

**IntEnable**(INT\_TIMER1A);

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

// Configure pins for UART

**GPIOPinConfigure**(GPIO\_PA0\_U0RX);

**GPIOPinConfigure**(GPIO\_PA1\_U0TX);

**GPIOPinTypeUART**(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1);

// Configure UART parameters

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

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

// Enable interrupts

**IntMasterEnable**();

**TimerEnable**(TIMER1\_BASE, TIMER\_A);

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

// Initial message to terminal display

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

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

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

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

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

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

**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');

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

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

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

**while** (1) // Wait forever

{

}

}

// To convert integer to character. This function is not in stdlib.h

**void** **itoa**(**long** **unsigned** **int** value, **char**\* result, **int** base)

{

**if** (base < 2 || base > 36) { \*result = '\0';}

**char**\* ptr = result, \*ptr1 = result, tmp\_char;

**int** tmp\_value;

**do** {

tmp\_value = value;

value /= base;

\*ptr++ = "zyxwvutsrqponmlkjihgfedcba9876543210123456789abcdefghijklmnopqrstuvwxyz" [35 + (tmp\_value - value \* base)];

} **while** ( value );

**if** (tmp\_value < 0) \*ptr++ = '-';

\*ptr-- = '\0';

**while**(ptr1 < ptr) {

tmp\_char = \*ptr;

\*ptr--= \*ptr1;

\*ptr1++ = tmp\_char;

}

}

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

**Task 02:**

Youtube Link: https://youtu.be/xmJSPcuvBl8

**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"

// Global variables

uint32\_t ui32ADC0Value[4];

**volatile** uint32\_t ui32TempAvg;

**volatile** uint32\_t ui32TempValueC;

**volatile** uint32\_t ui32TempValueF;

// Prototype function to convert int to char for UARTCharPut function

**void** **itoa**(**long** **unsigned** **int** value, **char**\* result, **int** base);

// UART isr

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

{

uint32\_t ui32Status;

uint32\_t command;

uint32\_t i;

**char** buffer[4];

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

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

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

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

**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;

itoa(ui32TempValueF, &buffer, 10); // converting integer to char

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

{

command = **UARTCharGet**(UART0\_BASE); // receiving character

**UARTCharPut**(UART0\_BASE, command); // echo character

**switch** (command)

{

**case** 'R': // Type R and Red LED turns on

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1 | GPIO\_PIN\_2 | GPIO\_PIN\_3 , 0);

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

**break**;

**case** 'B': // Type B and Blue LED turns on

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1 | GPIO\_PIN\_2 | GPIO\_PIN\_3 , 0);

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

**break**;

**case** 'G': // Type G and Green LED turns on

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1 | GPIO\_PIN\_2 | GPIO\_PIN\_3 , 0);

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

**break**;

**case** 'T': // Type T and temperature is displayed on terminal

**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');

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

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

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

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

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

**for** (i=0; i<4; i++)

{

**UARTCharPut**(UART0\_BASE, buffer[i] ); // temperature buffer to be displayed

}

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

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

**break**;

**case** 'r': // If any of these is typed, any LED on will turn off

**case** 'b':

**case** 'g':

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1 | GPIO\_PIN\_2 | GPIO\_PIN\_3 , 0);

**break**;

**default**:

**break**;

}

}

}

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

// Clock configuration

**SysCtlClockSet**(SYSCTL\_SYSDIV\_4 | SYSCTL\_USE\_PLL | SYSCTL\_OSC\_MAIN | SYSCTL\_XTAL\_16MHZ);

// Enable peripherals

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_UART0);

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

// Configure UART

**GPIOPinConfigure**(GPIO\_PA0\_U0RX);

**GPIOPinConfigure**(GPIO\_PA1\_U0TX);

**GPIOPinTypeUART**(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1);

// Enable F GPIOS for LEDs

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

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

// Enable ADC peripheral

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_ADC0);

**ADCHardwareOversampleConfigure**(ADC0\_BASE, 32);

**ADCSequenceConfigure**(ADC0\_BASE, 2, ADC\_TRIGGER\_PROCESSOR, 0); // Changed to sequencer #2

**ADCSequenceStepConfigure**(ADC0\_BASE, 2, 0, ADC\_CTL\_TS);

**ADCSequenceStepConfigure**(ADC0\_BASE, 2, 1, ADC\_CTL\_TS);

**ADCSequenceStepConfigure**(ADC0\_BASE, 2, 2, ADC\_CTL\_TS);

**ADCSequenceStepConfigure**(ADC0\_BASE, 2, 3, ADC\_CTL\_TS|ADC\_CTL\_IE|ADC\_CTL\_END);

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

// UART parameters

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

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

// Enable interrupts

**IntMasterEnable**();

**IntEnable**(INT\_UART0);

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

// Initial message

**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, 'r');

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

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

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

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

**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, 'o');

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

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

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

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

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

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

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

**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, 'o');

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

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

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

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

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

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

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

**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, ' ');

**while** (1)

{

}

}

// To convert integer to character. This function is not in stdlib.h

**void** **itoa**(**long** **unsigned** **int** value, **char**\* result, **int** base)

{

// check that the base if valid

**if** (base < 2 || base > 36) { \*result = '\0';}

**char**\* ptr = result, \*ptr1 = result, tmp\_char;

**int** tmp\_value;

**do** {

tmp\_value = value;

value /= base;

\*ptr++ = "zyxwvutsrqponmlkjihgfedcba9876543210123456789abcdefghijklmnopqrstuvwxyz" [35 + (tmp\_value - value \* base)];

} **while** ( value );

// Apply negative sign

**if** (tmp\_value < 0) \*ptr++ = '-';

\*ptr-- = '\0';

**while**(ptr1 < ptr) {

tmp\_char = \*ptr;

\*ptr--= \*ptr1;

\*ptr1++ = tmp\_char;

}

}