**Date Submitted: 10/12**

**Task 00: Execute provided code**

**The only modification made is adding the temperature interrupt from LAB 05.**

**Youtube Link:** [**https://youtu.be/CfjjsMAQCqs**](https://youtu.be/CfjjsMAQCqs)

**#include** <stdint.h>

**#include** <stdbool.h>

**#include** <stdlib.h>

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

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

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

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

**#define** TARGET\_IS\_BLIZZARD\_RA1

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

// Variables

uint32\_t ui32ADC0Value[4];

uint32\_t ui32Period;

**volatile** uint32\_t ui32TempAvg;

**volatile** uint32\_t ui32TempValueC;

**volatile** uint32\_t ui32TempValueF;

// Timer 1A ISR

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

{

// Reset the count on Timer1

ROM\_TimerDisable(TIMER1\_BASE, TIMER\_A); // Disable Timer1

ROM\_TimerLoadSet(TIMER1\_BASE, TIMER\_A, ui32Period); // (40 MHz / 0.5 Hz)

ROM\_TimerIntClear(TIMER1\_BASE, TIMER\_TIMA\_TIMEOUT); // Clear Timer1A flag

// Perform ADC conversion

ROM\_ADCIntClear(ADC0\_BASE, 1); // Clear ADC interrupt flag

ROM\_ADCProcessorTrigger(ADC0\_BASE, 1); // Trigger the ADC conversion

**while**(!ROM\_ADCIntStatus(ADC0\_BASE, 1, **false**)); // Poll until conversion completes

ROM\_ADCSequenceDataGet(ADC0\_BASE, 1, ui32ADC0Value); // Store the temperature value

ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] // Average the sampled temperatures

+ ui32ADC0Value[2] + ui32ADC0Value[3]

+ 2) / 4;

// Convert to Celsius

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

// Convert to Fahrenheit

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

ROM\_TimerEnable(TIMER1\_BASE, TIMER\_A); // Re-enable Timer1

}

**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);

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

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

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

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

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

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

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

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

// Enables the ADC

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_ADC0);

ROM\_ADCHardwareOversampleConfigure(ADC0\_BASE, 32); // Average at 32 samples

// ADC0, sample sequencer 1, processor triggers, highest priority

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);

// Set up TIMER1

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_TIMER1);

ROM\_TimerConfigure(TIMER1\_BASE, TIMER\_CFG\_PERIODIC);

// (50 MHz \* 0.5 Hz)

ui32Period = ROM\_SysCtlClockGet() \* 0.5;

ROM\_TimerLoadSet(TIMER1\_BASE, TIMER\_A, ui32Period - 1);

// Enable Timer1A interrupt

ROM\_IntEnable(INT\_TIMER1A);

ROM\_IntEnable(INT\_UART0);

ROM\_TimerIntEnable(TIMER1\_BASE, TIMER\_TIMA\_TIMEOUT);

ROM\_IntMasterEnable();

// Start TIMER1A

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

**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);

}

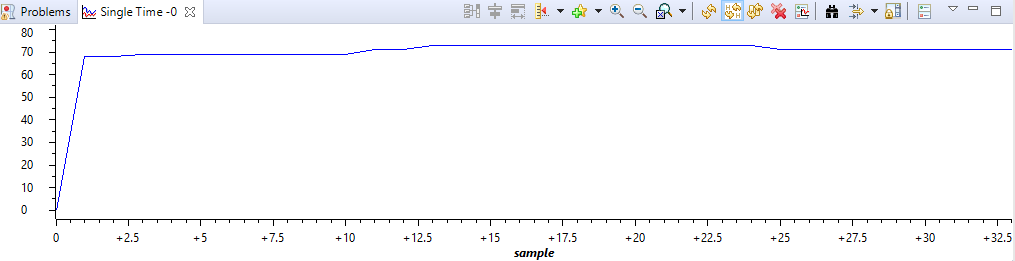
**------------------------------------------------------------------------------------**

**Task 01:**

Youtube Link: [**https://youtu.be/W2p1u-0rsF8**](https://youtu.be/W2p1u-0rsF8)

**Modified Code:**

**After adding the timer interrupt for the temperature from LAB 05, the printString function was added. This function prints a string of characters onto the terminal using UARTCharPut function. This is used to print “Temperature: ” and the measured temperature after it is converted using the ltoa function. For the ltoa to convert the temperature into a string, the char type “buffer” variable is needed to store the resulting string.**

****

**Image of the graph in CCS.**

**#include** <stdint.h>

**#include** <stdbool.h>

**#include** <stdlib.h>

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

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

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

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

**#define** TARGET\_IS\_BLIZZARD\_RA1

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

// Variables

uint32\_t ui32ADC0Value[4];

uint32\_t ui32Period;

**volatile** uint32\_t ui32TempAvg;

**volatile** uint32\_t ui32TempValueC;

**volatile** uint32\_t ui32TempValueF;

// Function to print a string to the terminal

**void** **printString**(**char** \*string)

{

**while**(\*string)

{

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

string++;

}

}

// Timer 1A ISR

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

{

**char** buffer[4];

// Reset the count on Timer1

ROM\_TimerDisable(TIMER1\_BASE, TIMER\_A); // Disable Timer1

ROM\_TimerLoadSet(TIMER1\_BASE, TIMER\_A, ui32Period); // (40 MHz / 0.5 Hz)

ROM\_TimerIntClear(TIMER1\_BASE, TIMER\_TIMA\_TIMEOUT); // Clear Timer1A flag

// Perform ADC conversion

ROM\_ADCIntClear(ADC0\_BASE, 1); // Clear ADC interrupt flag

ROM\_ADCProcessorTrigger(ADC0\_BASE, 1); // Trigger the ADC conversion

**while**(!ROM\_ADCIntStatus(ADC0\_BASE, 1, **false**)); // Poll until conversion completes

ROM\_ADCSequenceDataGet(ADC0\_BASE, 1, ui32ADC0Value); // Store the temperature value

ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] // Average the sampled temperatures

+ ui32ADC0Value[2] + ui32ADC0Value[3]

+ 2) / 4;

// Convert to Celsius

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

// Convert to Fahrenheit

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

**UARTCharPut**(UART0\_BASE, 0x0C); // Clear the terminal

**ltoa**(ui32TempValueF, buffer); // Convert the temperature into a string

// Print the temperature onto the terminal

printString("Temperature: ");

printString(buffer);

printString(" F");

// New line and return carriage

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

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

ROM\_TimerEnable(TIMER1\_BASE, TIMER\_A); // Re-enable Timer1

}

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

// f = 50 MHz

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

// Configure the UART, GPIOA, and GPIOF

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

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

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

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

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

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

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

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

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

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

// Enables the ADC

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_ADC0);

ROM\_ADCHardwareOversampleConfigure(ADC0\_BASE, 32); // Average at 32 samples

// ADC0, sample sequencer 1, processor triggers, highest priority

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);

// Set up TIMER1

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_TIMER1);

ROM\_TimerConfigure(TIMER1\_BASE, TIMER\_CFG\_PERIODIC);

// (50 MHz \* 0.5 Hz)

ui32Period = ROM\_SysCtlClockGet() \* 0.5;

ROM\_TimerLoadSet(TIMER1\_BASE, TIMER\_A, ui32Period - 1);

// Enable Timer1A interrupt

ROM\_IntEnable(INT\_TIMER1A);

ROM\_IntEnable(INT\_UART0);

ROM\_TimerIntEnable(TIMER1\_BASE, TIMER\_TIMA\_TIMEOUT);

ROM\_IntMasterEnable();

// Start TIMER1A

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

**while** (1);

}

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

**Task 02:**

Youtube Link: [**https://youtu.be/Mp0W1d\_qIag**](https://youtu.be/Mp0W1d_qIag)

**Modified Code:**

**Taking the code from task 01, the first modification is to add the UART interrupt back that was used in task 00. In the UART interrupt, there’s a switch statement used for checking commands such as ‘R’, ‘G’, etc. For the temperature to be toggled in the terminal, a global Boolean variable called “temp” is used to determine whether to display the temperature or not after timer1 interrupt converts the temperature. This Boolean “temp” value is also set in the switch statement [located in the UART interrupt].**

**#include** <stdint.h>

**#include** <stdbool.h>

**#include** <stdlib.h>

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

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

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

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

**#define** TARGET\_IS\_BLIZZARD\_RA1

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

// Variables

uint32\_t ui32ADC0Value[4];

uint32\_t ui32Period;

**volatile** uint32\_t ui32TempAvg;

**volatile** uint32\_t ui32TempValueC;

**volatile** uint32\_t ui32TempValueF;

**volatile** **bool** temp;

**char** buffer[4];

// Function to print a string to the terminal

**void** **printString**(**char** \*string)

{

**while**(\*string)

{

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

string++;

}

}

// Timer 1A ISR

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

{

// Reset the count on Timer1

ROM\_TimerDisable(TIMER1\_BASE, TIMER\_A); // Disable Timer1

ROM\_TimerLoadSet(TIMER1\_BASE, TIMER\_A, ui32Period); // (40 MHz / 0.5 Hz)

ROM\_TimerIntClear(TIMER1\_BASE, TIMER\_TIMA\_TIMEOUT); // Clear Timer1A flag

// Perform ADC conversion

ROM\_ADCIntClear(ADC0\_BASE, 1); // Clear ADC interrupt flag

ROM\_ADCProcessorTrigger(ADC0\_BASE, 1); // Trigger the ADC conversion

**while**(!ROM\_ADCIntStatus(ADC0\_BASE, 1, **false**)); // Poll until conversion completes

ROM\_ADCSequenceDataGet(ADC0\_BASE, 1, ui32ADC0Value); // Store the temperature value

ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] // Average the sampled temperatures

+ ui32ADC0Value[2] + ui32ADC0Value[3]

+ 2) / 4;

// Convert to Celsius

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

// Convert to Fahrenheit

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

**ltoa**(ui32TempValueF, buffer); // Convert the temperature into a string

// Check to display the temperature on the terminal

**if**(temp)

{

// Prints the temperature

**UARTCharPut**(UART0\_BASE, 0x0C); // Clear the terminal

printString("Temperature: ");

printString(buffer);

printString(" F");

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

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

}

**else**

**UARTCharPut**(UART0\_BASE, 0x0C); // Clear the terminal

ROM\_TimerEnable(TIMER1\_BASE, TIMER\_A); // Re-enable Timer1

}

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

{

**char** input;

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

{

input = **UARTCharGetNonBlocking**(UART0\_BASE); // Grab the inputs from the keyboard

// Check the inputs to execute the commands

**switch**(input)

{

// Red LED

**case** 'R':

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

**break**;

**case** 'r':

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

**break**;

// Blue LED

**case** 'B':

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

**break**;

**case** 'b':

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

**break**;

// Green LED

**case** 'G':

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

**break**;

**case** 'g':

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

**break**;

// Temperature

**case** 'T':

temp = **true**;

**break**;

**case** 't':

temp = **false**;

**break**;

}

}

}

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

// f = 50 MHz

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

// Configure the UART, GPIOA, and GPIOF

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

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

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

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

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

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

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

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

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

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

// Enables the ADC

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_ADC0);

ROM\_ADCHardwareOversampleConfigure(ADC0\_BASE, 32); // Average at 32 samples

// ADC0, sample sequencer 1, processor triggers, highest priority

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);

// Set up TIMER1

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_TIMER1);

ROM\_TimerConfigure(TIMER1\_BASE, TIMER\_CFG\_PERIODIC);

// (50 MHz \* 0.5 Hz)

ui32Period = ROM\_SysCtlClockGet() \* 0.5;

ROM\_TimerLoadSet(TIMER1\_BASE, TIMER\_A, ui32Period - 1);

// Enable Timer1A interrupt

ROM\_IntEnable(INT\_TIMER1A);

ROM\_IntEnable(INT\_UART0);

ROM\_TimerIntEnable(TIMER1\_BASE, TIMER\_TIMA\_TIMEOUT);

ROM\_IntMasterEnable();

// Start TIMER1A

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

**while** (1);

}

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