**Date Submitted:**

**Task 00: Execute provided code**

**Modified Code:**

**#include** <stdint.h>

**#include** <stdio.h>

**#include** <stdbool.h>

**#include** <stdlib.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"

//#define TARGET\_IS\_BLIZZARD\_RB1

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

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

**volatile** uint32\_t ui32TempAvg; //holds temperature average value

**volatile** uint32\_t ui32TempValueC; //holds temperature value in celsius

**volatile** uint32\_t ui32TempValueF; //holds temperature value in fahrenheit

uint32\_t ui32Period;

uint32\_t ui32ADC0Value[4]; //array holds data read

**char** str[10];

// Implementation of itoa()

**char**\* **itoa**( uint32\_t num, **char**\* str, **int** base)

{

**int** i = 0;

/\* Handle 0 explicitely, otherwise empty string is printed for 0 \*/

**if** (num == 0)

{

str[i++] = '0';

str[i] = '\0';

**return** str;

}

// Process individual digits

**while** (num != 0)

{

**int** rem = num % base;

str[i++] = (rem > 9)? (rem-10) + 'a' : rem + '0';

num = num/base;

}

str[i] = '\0'; // Append string terminator

**return** str;

}

**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); //set up clock

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_UART0); //enable UART0 peripheral

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOA); //enable GPIOA peripheral

**GPIOPinConfigure**(GPIO\_PA0\_U0RX); //configure pin as receiver

**GPIOPinConfigure**(GPIO\_PA1\_U0TX); //configure pin as transmitter

**GPIOPinTypeUART**(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1); //set pintype to UART

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOF); //initialize GPIO peripheral

**GPIOPinTypeGPIOOutput**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1|GPIO\_PIN\_2|GPIO\_PIN\_3); //enable GPIO pin for output for LED

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

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

//Initializes parameters for UART: 114200, 8-1-N

// IntEnable(INT\_UART0); //enable uart0 interrupt

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

/\*

UARTCharPut(UART0\_BASE, 'E'); //calls to create a prompt

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, ' '); //"Enter text: "

\*/

// str[0] = 'p';

// UARTCharPut(UART0\_BASE, str[0]);

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_ADC0); //enables ADC0 peripheral

ROM\_ADCHardwareOversampleConfigure(ADC0\_BASE, 64); //API call with 64 samples to be averaged

ROM\_ADCSequenceConfigure(ADC0\_BASE, 1, ADC\_TRIGGER\_PROCESSOR, 0); //use ADC0, sample sequencer 1

//want the processor to trigger sequence and want to use highest priority

ROM\_ADCSequenceStepConfigure(ADC0\_BASE, 1, 0, ADC\_CTL\_TS); //first step of adc sequencer

ROM\_ADCSequenceStepConfigure(ADC0\_BASE, 1, 1, ADC\_CTL\_TS); //second step

ROM\_ADCSequenceStepConfigure(ADC0\_BASE, 1, 2, ADC\_CTL\_TS); //third step

ROM\_ADCSequenceStepConfigure(ADC0\_BASE,1,3,ADC\_CTL\_TS|ADC\_CTL\_IE|ADC\_CTL\_END); //fourth step will

//configure interrupt flag, sample the temperature sensor, and tell adc logic that this

//is the last step

ROM\_ADCSequenceEnable(ADC0\_BASE, 1); //enable ADC sequencer 1

ROM\_ADCIntClear(ADC0\_BASE, 1); //clear interrupt status flag

ROM\_ADCProcessorTrigger(ADC0\_BASE, 1); //trigger adc conversion

**while**(!ROM\_ADCIntStatus(ADC0\_BASE, 1, false)) //wait for conversion to finish

{

}

ROM\_ADCSequenceDataGet(ADC0\_BASE, 1, ui32ADC0Value); //copies data from ...

//the specified saple sequencer output FIFO to a buffer in memory

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

//calculate the average temperature value

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

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

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_TIMER0); //enable timer

**TimerConfigure**(TIMER0\_BASE, TIMER\_CFG\_PERIODIC); //configure timer 0

**IntEnable**(INT\_TIMER0A); //enable timer interrupt

**TimerIntEnable**(TIMER0\_BASE, TIMER\_TIMA\_TIMEOUT);

**IntMasterEnable**(); //enable all interrupts

ui32Period = (**SysCtlClockGet**() / 10) \* 5; //0.5 ms period

**TimerLoadSet**(TIMER0\_BASE, TIMER\_A, ui32Period -1);

**TimerEnable**(TIMER0\_BASE, TIMER\_A);

**while** (1) //infinite loop

{

//echoes what is types in terminal

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

}

}

**void** **Timer0IntHandler**(**void**)

{

**if**(**GPIOPinRead**(GPIO\_PORTF\_BASE, GPIO\_PIN\_2))

{

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

}

**else**

{

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_2, 4); //turn on led

}

// Clear the timer interrupt

**TimerIntClear**(TIMER0\_BASE, TIMER\_TIMA\_TIMEOUT);

ROM\_ADCIntClear(ADC0\_BASE, 1); //clear interrupt status flag

ROM\_ADCProcessorTrigger(ADC0\_BASE, 1); //trigger adc conversion

**while**(!ROM\_ADCIntStatus(ADC0\_BASE, 1, false)) //wait for conversion to finish

{

}

ROM\_ADCSequenceDataGet(ADC0\_BASE, 1, ui32ADC0Value); //copies data from ...

//the specified sample sequencer output FIFO to a buffer in memory

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

//calculate the average temperature value

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

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

itoa(ui32TempValueF, str, 10);

**UARTCharPut**(UART0\_BASE, str[1]);

**UARTCharPut**(UART0\_BASE, str[0]);

**UARTCharPut**(UART0\_BASE, '\r'); //carriage return

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

;

}

**Youtube Link:**

<https://youtu.be/xu-XoFlhbhE>

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

**Task 01:**

Youtube Link:

<https://youtu.be/5HC2qxpemiA>

**Modified Code:**

**#include** <stdint.h>

**#include** <stdio.h>

**#include** <stdbool.h>

**#include** <stdlib.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"

//#define TARGET\_IS\_BLIZZARD\_RB1

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

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

**volatile** uint32\_t ui32TempAvg; //holds temperature average value

**volatile** uint32\_t ui32TempValueC; //holds temperature value in celsius

**volatile** uint32\_t ui32TempValueF; //holds temperature value in fahrenheit

uint32\_t ui32Period;

uint32\_t ui32ADC0Value[4]; //array holds data read

**char** str[10];

// Implementation of itoa()

**char**\* **itoa**( uint32\_t num, **char**\* str, **int** base)

{

**int** i = 0;

/\* Handle 0 explicitely, otherwise empty string is printed for 0 \*/

**if** (num == 0)

{

str[i++] = '0';

str[i] = '\0';

**return** str;

}

// Process individual digits

**while** (num != 0)

{

**int** rem = num % base;

str[i++] = (rem > 9)? (rem-10) + 'a' : rem + '0';

num = num/base;

}

str[i] = '\0'; // Append string terminator

**return** str;

}

**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); //set up clock

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_UART0); //enable UART0 peripheral

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOA); //enable GPIOA peripheral

**GPIOPinConfigure**(GPIO\_PA0\_U0RX); //configure pin as receiver

**GPIOPinConfigure**(GPIO\_PA1\_U0TX); //configure pin as transmitter

**GPIOPinTypeUART**(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1); //set pintype to UART

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOF); //initialize GPIO peripheral

**GPIOPinTypeGPIOOutput**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1|GPIO\_PIN\_2|GPIO\_PIN\_3); //enable GPIO pin for output for LED

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

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

//Initializes parameters for UART: 114200, 8-1-N

// IntEnable(INT\_UART0); //enable uart0 interrupt

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

/\*

UARTCharPut(UART0\_BASE, 'E'); //calls to create a prompt

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, ' '); //"Enter text: "

\*/

// str[0] = 'p';

// UARTCharPut(UART0\_BASE, str[0]);

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_ADC0); //enables ADC0 peripheral

ROM\_ADCHardwareOversampleConfigure(ADC0\_BASE, 64); //API call with 64 samples to be averaged

ROM\_ADCSequenceConfigure(ADC0\_BASE, 1, ADC\_TRIGGER\_PROCESSOR, 0); //use ADC0, sample sequencer 1

//want the processor to trigger sequence and want to use highest priority

ROM\_ADCSequenceStepConfigure(ADC0\_BASE, 1, 0, ADC\_CTL\_TS); //first step of adc sequencer

ROM\_ADCSequenceStepConfigure(ADC0\_BASE, 1, 1, ADC\_CTL\_TS); //second step

ROM\_ADCSequenceStepConfigure(ADC0\_BASE, 1, 2, ADC\_CTL\_TS); //third step

ROM\_ADCSequenceStepConfigure(ADC0\_BASE,1,3,ADC\_CTL\_TS|ADC\_CTL\_IE|ADC\_CTL\_END); //fourth step will

//configure interrupt flag, sample the temperature sensor, and tell adc logic that this

//is the last step

ROM\_ADCSequenceEnable(ADC0\_BASE, 1); //enable ADC sequencer 1

ROM\_ADCIntClear(ADC0\_BASE, 1); //clear interrupt status flag

ROM\_ADCProcessorTrigger(ADC0\_BASE, 1); //trigger adc conversion

**while**(!ROM\_ADCIntStatus(ADC0\_BASE, 1, false)) //wait for conversion to finish

{

}

ROM\_ADCSequenceDataGet(ADC0\_BASE, 1, ui32ADC0Value); //copies data from ...

//the specified saple sequencer output FIFO to a buffer in memory

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

//calculate the average temperature value

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

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

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_TIMER0); //enable timer

**TimerConfigure**(TIMER0\_BASE, TIMER\_CFG\_PERIODIC); //configure timer 0

**IntEnable**(INT\_TIMER0A); //enable timer interrupt

**TimerIntEnable**(TIMER0\_BASE, TIMER\_TIMA\_TIMEOUT);

**IntMasterEnable**(); //enable all interrupts

ui32Period = (**SysCtlClockGet**() / 10) \* 5; //0.5 ms period

**TimerLoadSet**(TIMER0\_BASE, TIMER\_A, ui32Period -1);

**TimerEnable**(TIMER0\_BASE, TIMER\_A);

**while** (1) //infinite loop

{

//echoes what is types in terminal

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

}

}

**void** **Timer0IntHandler**(**void**)

{

**if**(**GPIOPinRead**(GPIO\_PORTF\_BASE, GPIO\_PIN\_2))

{

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

}

**else**

{

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_2, 4); //turn on led

}

// Clear the timer interrupt

**TimerIntClear**(TIMER0\_BASE, TIMER\_TIMA\_TIMEOUT);

ROM\_ADCIntClear(ADC0\_BASE, 1); //clear interrupt status flag

ROM\_ADCProcessorTrigger(ADC0\_BASE, 1); //trigger adc conversion

**while**(!ROM\_ADCIntStatus(ADC0\_BASE, 1, false)) //wait for conversion to finish

{

}

ROM\_ADCSequenceDataGet(ADC0\_BASE, 1, ui32ADC0Value); //copies data from ...

//the specified sample sequencer output FIFO to a buffer in memory

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

//calculate the average temperature value

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

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

itoa(ui32TempValueF, str, 10);

**UARTCharPut**(UART0\_BASE, str[1]);

**UARTCharPut**(UART0\_BASE, str[0]);

**UARTCharPut**(UART0\_BASE, '\r'); //carriage return

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

;

}

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

**Task 02:**

Youtube Link:

<https://youtu.be/fPus-2z6V3c>

**Modified Code:**

**#include** <stdint.h>

**#include** <stdio.h>

**#include** <stdbool.h>

**#include** <stdlib.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"

//#define TARGET\_IS\_BLIZZARD\_RB1

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

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

**volatile** uint32\_t ui32TempAvg; //holds temperature average value

**volatile** uint32\_t ui32TempValueC; //holds temperature value in celsius

**volatile** uint32\_t ui32TempValueF; //holds temperature value in fahrenheit

uint32\_t ui32Period;

uint32\_t ui32ADC0Value[4]; //array holds data read

**char** str[10];

// Implementation of itoa()

**char**\* **itoa**( uint32\_t num, **char**\* str, **int** base)

{

**int** i = 0;

/\* Handle 0 explicitely, otherwise empty string is printed for 0 \*/

**if** (num == 0)

{

str[i++] = '0';

str[i] = '\0';

**return** str;

}

// Process individual digits

**while** (num != 0)

{

**int** rem = num % base;

str[i++] = (rem > 9)? (rem-10) + 'a' : rem + '0';

num = num/base;

}

str[i] = '\0'; // Append string terminator

**return** str;

}

**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); //set up clock

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_UART0); //enable UART0 peripheral

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOA); //enable GPIOA peripheral

**GPIOPinConfigure**(GPIO\_PA0\_U0RX); //configure pin as receiver

**GPIOPinConfigure**(GPIO\_PA1\_U0TX); //configure pin as transmitter

**GPIOPinTypeUART**(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1); //set pintype to UART

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOF); //initialize GPIO peripheral

**GPIOPinTypeGPIOOutput**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1|GPIO\_PIN\_2|GPIO\_PIN\_3); //enable GPIO pin for output for LED

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

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

//Initializes parameters for UART: 114200, 8-1-N

// IntEnable(INT\_UART0); //enable uart0 interrupt

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

/\*

UARTCharPut(UART0\_BASE, 'E'); //calls to create a prompt

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, ' '); //"Enter text: "

\*/

// str[0] = 'p';

// UARTCharPut(UART0\_BASE, str[0]);

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_ADC0); //enables ADC0 peripheral

ROM\_ADCHardwareOversampleConfigure(ADC0\_BASE, 64); //API call with 64 samples to be averaged

ROM\_ADCSequenceConfigure(ADC0\_BASE, 1, ADC\_TRIGGER\_PROCESSOR, 0); //use ADC0, sample sequencer 1

//want the processor to trigger sequence and want to use highest priority

ROM\_ADCSequenceStepConfigure(ADC0\_BASE, 1, 0, ADC\_CTL\_TS); //first step of adc sequencer

ROM\_ADCSequenceStepConfigure(ADC0\_BASE, 1, 1, ADC\_CTL\_TS); //second step

ROM\_ADCSequenceStepConfigure(ADC0\_BASE, 1, 2, ADC\_CTL\_TS); //third step

ROM\_ADCSequenceStepConfigure(ADC0\_BASE,1,3,ADC\_CTL\_TS|ADC\_CTL\_IE|ADC\_CTL\_END); //fourth step will

//configure interrupt flag, sample the temperature sensor, and tell adc logic that this

//is the last step

ROM\_ADCSequenceEnable(ADC0\_BASE, 1); //enable ADC sequencer 1

ROM\_ADCIntClear(ADC0\_BASE, 1); //clear interrupt status flag

ROM\_ADCProcessorTrigger(ADC0\_BASE, 1); //trigger adc conversion

**while**(!ROM\_ADCIntStatus(ADC0\_BASE, 1, false)) //wait for conversion to finish

{

}

ROM\_ADCSequenceDataGet(ADC0\_BASE, 1, ui32ADC0Value); //copies data from ...

//the specified saple sequencer output FIFO to a buffer in memory

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

//calculate the average temperature value

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

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

/\*

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_TIMER0); //enable timer

TimerConfigure(TIMER0\_BASE, TIMER\_CFG\_PERIODIC); //configure timer 0

IntEnable(INT\_TIMER0A); //enable timer interrupt

TimerIntEnable(TIMER0\_BASE, TIMER\_TIMA\_TIMEOUT);

IntMasterEnable(); //enable all interrupts

ui32Period = (SysCtlClockGet() / 10) \* 0.1; //0.5 s period

TimerLoadSet(TIMER0\_BASE, TIMER\_A, ui32Period -1);

TimerEnable(TIMER0\_BASE, TIMER\_A);

\*/

**while** (1) //infinite loop

{

ROM\_ADCIntClear(ADC0\_BASE, 1); //clear interrupt status flag

ROM\_ADCProcessorTrigger(ADC0\_BASE, 1); //trigger adc conversion

**while**(!ROM\_ADCIntStatus(ADC0\_BASE, 1, false)) //wait for conversion to finish

{

}

ROM\_ADCSequenceDataGet(ADC0\_BASE, 1, ui32ADC0Value); //copies data from ...

//the specified sample sequencer output FIFO to a buffer in memory

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

//calculate the average temperature value

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

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

itoa(ui32TempValueF, str, 10);

**if** (**UARTCharGet**(UART0\_BASE) == 'T'){ //check temperature

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

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

**UARTCharPut**(UART0\_BASE, str[1]);

**UARTCharPut**(UART0\_BASE, str[0]);

**UARTCharPut**(UART0\_BASE, '\r'); //carriage return

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

}

**if** (**UARTCharGet**(UART0\_BASE) == 'R'){

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

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1, 2); //turn on red led

**UARTCharPut**(UART0\_BASE, '\r'); //carriage return

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

}

**if** (**UARTCharGet**(UART0\_BASE) == 'r'){

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

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

**UARTCharPut**(UART0\_BASE, '\r'); //carriage return

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

}

**if** (**UARTCharGet**(UART0\_BASE) == 'B') {

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

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_2, 4); //turn on blue led

**UARTCharPut**(UART0\_BASE, '\r'); //carriage return

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

}

**if** (**UARTCharGet**(UART0\_BASE) == 'b') {

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

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

**UARTCharPut**(UART0\_BASE, '\r'); //carriage return

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

}

**if** (**UARTCharGet**(UART0\_BASE) == 'G') {

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

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_3, 8); //turn on green led

**UARTCharPut**(UART0\_BASE, '\r'); //carriage return

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

}

**if** (**UARTCharGet**(UART0\_BASE) == 'g') {

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

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_3, 0); //turn off green led

**UARTCharPut**(UART0\_BASE, '\r'); //carriage return

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

}

}

}

/\*

void Timer0IntHandler(void)

{

if(GPIOPinRead(GPIO\_PORTF\_BASE, GPIO\_PIN\_2))

{

GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_1|GPIO\_PIN\_2|GPIO\_PIN\_3, 0); //turn off led

}

else

{

GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_2, 4); //turn on led

}

// Clear the timer interrupt

TimerIntClear(TIMER0\_BASE, TIMER\_TIMA\_TIMEOUT);

ROM\_ADCIntClear(ADC0\_BASE, 1); //clear interrupt status flag

ROM\_ADCProcessorTrigger(ADC0\_BASE, 1); //trigger adc conversion

while(!ROM\_ADCIntStatus(ADC0\_BASE, 1, false)) //wait for conversion to finish

{

}

ROM\_ADCSequenceDataGet(ADC0\_BASE, 1, ui32ADC0Value); //copies data from ...

//the specified sample sequencer output FIFO to a buffer in memory

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

//calculate the average temperature value

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

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

itoa(ui32TempValueF, str, 10);

if (UARTCharGet(UART0\_BASE) == 'T'){ //check temperature

UARTCharPut(UART0\_BASE, UARTCharGet(UART0\_BASE));

UARTCharPut(UART0\_BASE, str[1]);

UARTCharPut(UART0\_BASE, str[0]);

UARTCharPut(UART0\_BASE, '\r'); //carriage return

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

}

}

\*/

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