**Date Submitted: 10/14/2019**

**Task 00: Execute provided code**

**Youtube Link:** <https://www.youtube.com/watch?v=l9ba81s9mzI>

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

**Task 01:**

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

**Modified Schematic (if applicable):**

**Modified Code:**

**#include** <stdint.h>

**#include** <stdbool.h>

**#include** <stdio.h>

**#include** <stdlib.h>

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

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

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

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

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

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

**#include** "driverlib/pin\_map.h"

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

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

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

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

//global variables

uint32\_t ui32ADC0Value[4]; // array to store samples of ADC with 4 steps

**volatile** uint32\_t ui32TempAvg; // stores avg temp

**volatile** uint32\_t ui32TempValueC; // stores temp in Celsius

**volatile** uint32\_t ui32TempValueF; // stores temp in Fahrenheit

**char** str\_temp[10]; // variable used to store temp value in string

//print function

**void** **print\_string**(**char** \* str) {

**while**(\*str != '\0')

{

**UARTCharPut**(UART0\_BASE,\*str); // print temp value on terminal

++str;

}

}

//reverse() function in association with itoa function

//reverses string

**void** **reverse**(**char** str[], **int** len)

{

**int** start, end;

**char** temp;

**for**(start=0, end=len-1; start < end; start++, end--) {

temp = \*(str+start);

\*(str+start) = \*(str+end);

\*(str+end) = temp;

}

}

//itoa() function to convert from int to string to display on terminal

//returns the converted temp value in string

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

{

**int** i = 0;

**bool** isNegative = **false**;

/\* A zero is same "0" string in all base \*/

**if** (num == 0) {

str[i] = '0';

str[i + 1] = '\0';

**return** str;

}

/\* negative numbers are only handled if base is 10

otherwise considered unsigned number \*/

**if** (num < 0 && base == 10) {

isNegative = **true**;

num = -num;

}

**while** (num != 0) {

**int** rem = num % base;

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

num = num/base;

}

/\* Append negative sign for negative numbers \*/

**if** (isNegative){

str[i++] = '-';

}

str[i] = '\0';

reverse(str, i);

**return** str;

}

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

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

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_UART0); //enable uart0 peripheral

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOA); //UART pins located on GPIO PORTA

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_ADC0); //enable ADC0 peripheral

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

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

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

**GPIOPinTypeUART**(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1); //configure pins as UART

//set up UART: using system clk, baud rate: 115200, 8 data bits, 1 stop bit, and no parity bits

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

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

//Configure ADC

**ADCSequenceConfigure**(ADC0\_BASE, 1, ADC\_TRIGGER\_PROCESSOR, 0); // using ADC sample sequencer 1 (SS1), set as the highest priority, and processor will trigger ADC

**ADCSequenceStepConfigure**(ADC0\_BASE, 1, 0, ADC\_CTL\_TS); // ADC sample step 0

**ADCSequenceStepConfigure**(ADC0\_BASE, 1, 1, ADC\_CTL\_TS); // ADC sample step 1

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

**ADCSequenceStepConfigure**(ADC0\_BASE,1,3,ADC\_CTL\_TS|ADC\_CTL\_IE|ADC\_CTL\_END); //ADC sample step 3, set ADC interrupt flag, end sampling

**ADCSequenceEnable**(ADC0\_BASE, 1); // enable ADC0

//configure global interrupt and timer1 interrupt

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

**IntEnable**(INT\_TIMER1A); //enables timer1A interrupt in the interrupt vector table

TimerIntEnable(TIMER1\_BASE, TIMER\_TIMA\_TIMEOUT); //interrupt is triggered at TIMEOUT of timer1A

//Configure Timer1

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

TimerEnable(TIMER1\_BASE, TIMER\_A);

TimerLoadSet(TIMER1\_BASE, TIMER\_A, **SysCtlClockGet**()/2); //period (duty cycle) of 0.5s

**while** (1) //let interrupt handler do the UART echo function

{

//wait for interrupt

}

}

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

// Clear the timer interrupt

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

**ADCIntClear**(ADC0\_BASE, 1); // clear ADC interrupt

**ADCProcessorTrigger**(ADC0\_BASE, 1); // processor begins to trigger ADC

**while**(!**ADCIntStatus**(ADC0\_BASE, 1, **false**)) // wait for ADC conversion..

{

}

**ADCSequenceDataGet**(ADC0\_BASE, 1, ui32ADC0Value); // get ADC value from samples

//calculations of temperatures

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

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

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

//convert temperature to string and display on terminal (only in Fahrenheit)

print\_string(itoa(ui32TempValueF, str\_temp, 10));

print\_string("\r\n"); //carriage return and line feed to current and previous temp values

}

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

**Task 02:**

Youtube Link: <https://www.youtube.com/watch?v=4cAePOMpJkU>

**Modified Schematic (if applicable):**

**Modified Code:**

**#include** <stdint.h>

**#include** <stdbool.h>

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** <string.h>

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

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

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

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

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

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

**#include** "driverlib/pin\_map.h"

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

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

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

//global variables

uint32\_t ui32ADC0Value[4]; // array to store samples of ADC with 4 steps

**volatile** uint32\_t ui32TempAvg; // stores avg temp

**volatile** uint32\_t ui32TempValueC; // stores temp in Celsius

**volatile** uint32\_t ui32TempValueF; // stores temp in Fahrenheit

**char** str\_temp[10]; // variable used to store temp value in string

**char** command; //character representing the command to perform

//print function

**void** **print\_string**(**char** \* str) {

**while**(\*str != '\0')

{

**UARTCharPut**(UART0\_BASE,\*str); // print temp value on terminal

++str;

}

}

//itoa() function to convert from int to string to display on terminal

//returns the converted temp value in string

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

{

**int** i = 0;

**bool** isNegative = **false**;

/\* A zero is same "0" string in all base \*/

**if** (num == 0) {

str[i] = '0';

str[i + 1] = '\0';

**return** str;

}

/\* negative numbers are only handled if base is 10

otherwise considered unsigned number \*/

**if** (num < 0 && base == 10) {

isNegative = **true**;

num = -num;

}

**while** (num != 0) {

**int** rem = num % base;

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

num = num/base;

}

/\* Append negative sign for negative numbers \*/

**if** (isNegative){

str[i++] = '-';

}

str[i] = '\0';

//reverse the temperature string (swap the 2 numbers)

**char** temp;

**int** i1 = 0;

**int** i2 = i-1; //length of string - 1

temp = str[i1];

str[i1] = str[i2];

str[i2] = temp;

**return** str;

}

**void** **prompt**(**void**) {

**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, ' ');

**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, ' ');

**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, ' ');

**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, '\n');

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

}

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

{

uint32\_t ui32Status;

ui32Status = **UARTIntStatus**(UART0\_BASE, **true**); //get interrupt status - using RX and TX interrupts only

**UARTIntClear**(UART0\_BASE, ui32Status); //clear the asserted interrupts

**while**(**UARTCharsAvail**(UART0\_BASE)) //loop while there are chars

{

command = **UARTCharGet**(UART0\_BASE); //obtain&store char command

**UARTCharPut**(UART0\_BASE, command); //display the char command

**switch**(command) {

**case** 'R':

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

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

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

**break**;

**case** 'B':

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

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

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

**break**;

**case** 'G':

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

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

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

**break**;

**case** 'T':

**ADCIntClear**(ADC0\_BASE, 1); // clear ADC interrupt

**ADCProcessorTrigger**(ADC0\_BASE, 1); // processor begins to trigger ADC

**while**(!**ADCIntStatus**(ADC0\_BASE, 1, **false**)) // wait for ADC conversion..

{

}

**ADCSequenceDataGet**(ADC0\_BASE, 1, ui32ADC0Value); // get ADC value from samples

//calculations of temperatures

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

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

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

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

itoa(ui32TempValueF, str\_temp, 10);

print\_string(str\_temp);

print\_string("F\n"); //carriage return and line feed to current and previous temp values

**break**;

**default**:

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

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

}

prompt();

}

}

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

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

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

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOA); //UART pins located on GPIO PORTA

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_ADC0); //enable ADC0 peripheral

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

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

**GPIOPinTypeUART**(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1); //configure pins as UART

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOF); //enable GPIO port for LED

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

//Configure ADC

**ADCSequenceConfigure**(ADC0\_BASE, 1, ADC\_TRIGGER\_PROCESSOR, 0); // using ADC sample sequencer 1 (SS1), set as the highest priority, and processor will trigger ADC

**ADCSequenceStepConfigure**(ADC0\_BASE, 1, 0, ADC\_CTL\_TS); // ADC sample step 0

**ADCSequenceStepConfigure**(ADC0\_BASE, 1, 1, ADC\_CTL\_TS); // ADC sample step 1

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

**ADCSequenceStepConfigure**(ADC0\_BASE,1,3,ADC\_CTL\_TS|ADC\_CTL\_IE|ADC\_CTL\_END); //ADC sample step 3, set ADC interrupt flag, end sampling

**ADCSequenceEnable**(ADC0\_BASE, 1); // enable ADC0

//set up UART: using system clk, baud rate: 115200, 8 data bits, 1 stop bit, and no parity bits

**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

//prompt "R: red, G: green, B: blue, T: temperature:"

prompt();

**while** (1) //let interrupt handler do the UART echo function

{

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

}

}

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