Date Submitted: 10/13/2019

**Task 00: Execute provided code**

**Youtube Link: https://youtu.be/iXFIWlzbr9UA screenshot of a social media post

Description automatically generated**

This is the same graph from the YouTube video. The temperatures are being sampled every 0.5 seconds. Watch the full YouTube video to see a zoom in on the temperature values. The cup was cold which brought the temperature down. I then rubbed the device to create heat in the form of friction to bring the temperature back up.

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

**Task 01:**

Youtube Link:

Lab7-Task01ccsGUI: <https://youtu.be/l3fbZAdFhe8>

Lab7-Task01putty: <https://youtu.be/bBbYXPPHRp4>

**TEMPERATURES ARE SAMPLED AT 0.5 SECONDS USING THE TIMER 1 INTERRUPT. I FORGOT TO MENTION THIS IN THE VIDEOS BUT IT IS EVIDENT BY THE GRAPH ABOVE AND THE CODE I CREATED.**

**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" //needed for timer 1 interrupt

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

**#include** "inc/tm4c123gh6pm.h"

**#include**"driverlib/interrupt.h" //needed for interrupt functions

**#include** "driverlib/timer.h" //needed for timer functions

**#ifdef** DEBUG

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

{

}

**#endif**

uint32\_t ui32ADC0Value[1]; //value of 1 for adc0 step 3

**volatile** uint32\_t ui32TempAvg;

**volatile** uint32\_t ui32TempValueC;

**volatile** uint32\_t ui32TempValueF;

//Function: 8 bit ascii character to be transferred

**void** **UartIntToChar** (**char** x)

{

**while**((UART0\_FR\_R&UART\_FR\_TXFF) != 0);

UART0\_DR\_R = x;

}

//Function:Use recursion to convert a 32 bit integer to a string

**void** **UartTransmit** (uint32\_t x)

{

**if** (x>=10)

{

UartTransmit(x / 10);

x = x % 10;

}

UartIntToChar(x+'0');

}

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

**SysCtlClockSet**(SYSCTL\_SYSDIV\_5 | SYSCTL\_USE\_PLL | SYSCTL\_OSC\_MAIN | SYSCTL\_XTAL\_16MHZ);

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

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

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

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

//Uart Configure

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

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

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

//ADC configure

**ADCSequenceConfigure**(ADC0\_BASE, 3, ADC\_TRIGGER\_PROCESSOR, 0);

**ADCSequenceStepConfigure**(ADC0\_BASE,3,0,ADC\_CTL\_TS|ADC\_CTL\_IE|ADC\_CTL\_END);

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

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

//UART configure

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

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

//Timer 1 Configure

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_TIMER1);

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

**TimerLoadSet**(TIMER1\_BASE, TIMER\_A, (**SysCtlClockGet**() \* .5));

**IntEnable**(INT\_TIMER1A);

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

**IntMasterEnable**();

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

**ADCSequenceEnable**(ADC0\_BASE, 3);

**ADCIntEnable**(ADC0\_BASE, 3); //enable interrupt for sequence 3

**while** (1)

{

}

}

**void** **Timer1IntHandler**(**void**) //created timer1 interrupt handler

{

**ADCIntClear**(ADC0\_BASE, 3); //clear adc conversion done flag before writing code that depends on it. change to sequence 2

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

**TimerLoadSet**(TIMER1\_BASE, TIMER\_A, (**SysCtlClockGet**() \* .5)); //timer loaded to .5 seconds

**ADCProcessorTrigger**(ADC0\_BASE, 3); //changed to sequence 3

**while**(!**ADCIntStatus**(ADC0\_BASE, 3, **false**)) //wait for conversion to finish

{

} //if loop exited conversion is complete

**ADCSequenceDataGet**(ADC0\_BASE, 3, ui32ADC0Value); //gets samples from the array

ui32TempValueC = (1475 -((2475 \* ui32ADC0Value[0])) / 4096)/10; //only one adcval

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

UartTransmit(ui32TempValueF); //send the temperature to be converted to a string

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

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

}

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

**Task 02:**

Youtube Link: https://youtu.be/L8viAwgOLTc

**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" //needed for interrupts

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

**#include** "inc/tm4c123gh6pm.h"

**#include**"driverlib/interrupt.h" //needed for interrupt functions

**#ifdef** DEBUG

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

{

}

**#endif**

uint32\_t ui32ADC0Value[1]; //value of 1 for adc0 step 3

**volatile** uint32\_t ui32TempAvg;

**volatile** uint32\_t ui32TempValueC;

**volatile** uint32\_t ui32TempValueF;

//Function: 8 bit ascii character to be transferred

**void** **UartIntToChar** (**char** x)

{

**while**((UART0\_FR\_R&UART\_FR\_TXFF) != 0);

UART0\_DR\_R = x;

}

//Function:Use recursion to convert a 32 bit integer to a string

**void** **UartTransmit** (uint32\_t x)

{

**if** (x>=10)

{

UartTransmit(x / 10);

x = x % 10;

}

UartIntToChar(x+'0');

}

**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

{

**switch**(**UARTCharGet**(UART0\_BASE)) //read the UART character that has been retreived

{

**case** 'R': //turn on red led

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

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

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

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

**SysCtlDelay**(**SysCtlClockGet**() / (1000\*3)); //key debouncing

**break**;

**case** 'r': //turn off red led

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

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

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

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

**SysCtlDelay**(**SysCtlClockGet**() / (1000\*3)); //key debouncing

**break**;

**case** 'B': //turn on blue led

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

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

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

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

**SysCtlDelay**(**SysCtlClockGet**() / (1000\*3)); //key debouncing

**break**;

**case** 'b': //turn off blue led

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

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

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

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

**SysCtlDelay**(**SysCtlClockGet**() / (1000\*3)); //key debouncing

**break**;

**case** 'G': //turn on green led

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

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

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

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

**SysCtlDelay**(**SysCtlClockGet**() / (1000\*3)); //key debouncing

**break**;

**case** 'g': //turn off green led

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

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

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

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

**SysCtlDelay**(**SysCtlClockGet**() / (1000\*3)); //key debouncing

**break**;

**case** 'T': //calc and display temp

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

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

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

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

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

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

UartTransmit (ui32TempValueF); //send temp to the transmit function to display in terminal

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

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

**SysCtlDelay**(**SysCtlClockGet**() / (1000\*3)); //key debouncing

**break**;

}

}

}

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

**SysCtlClockSet**(SYSCTL\_SYSDIV\_5 | SYSCTL\_USE\_PLL | SYSCTL\_OSC\_MAIN | SYSCTL\_XTAL\_16MHZ);

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

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

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

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

//Uart Configure

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

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

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

//ADC configure

**ADCSequenceConfigure**(ADC0\_BASE, 3, ADC\_TRIGGER\_PROCESSOR, 0);

**ADCSequenceStepConfigure**(ADC0\_BASE,3,0,ADC\_CTL\_TS|ADC\_CTL\_IE|ADC\_CTL\_END);

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

**GPIOPinTypeGPIOOutput**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1|GPIO\_PIN\_2|GPIO\_PIN\_3); //enable pin for LED PF2

//UART configure

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

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

**ADCSequenceEnable**(ADC0\_BASE, 3);

**ADCIntEnable**(ADC0\_BASE, 3); //enable interrupt for sequence 3

**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

**while** (1)

{

**ADCIntClear**(ADC0\_BASE, 3); //clear adc conversion done flag before writing code that depends on it

**ADCProcessorTrigger**(ADC0\_BASE, 3); //changed to sequence 3

**while**(!**ADCIntStatus**(ADC0\_BASE, 3, **false**)) //wait for conversion to finish

{

} //if loop exited conversion is complete

**ADCSequenceDataGet**(ADC0\_BASE, 3, ui32ADC0Value); //gets samples from the array

ui32TempValueC = (1475 -((2475 \* ui32ADC0Value[0])) / 4096)/10; //only one adcval

ui32TempValueF = ((ui32TempValueC \* 9) + 160) / 5; //continuously calculate the temperature

}

}