Lab 9 Temperature Data Acquisition System

Duc Tran & Brandon Wong
11/17/2013

1.0 Objectives

The objectives of this lab is to study Analog to digital conversion. Student applied Nyquist theorem and valvano postulate to determine appropriate sampling rate for the temperature data acquisition system.

2.0 Hardware Design

2.1 For this temperature Data acquisition system, we used a 2.5V reference.

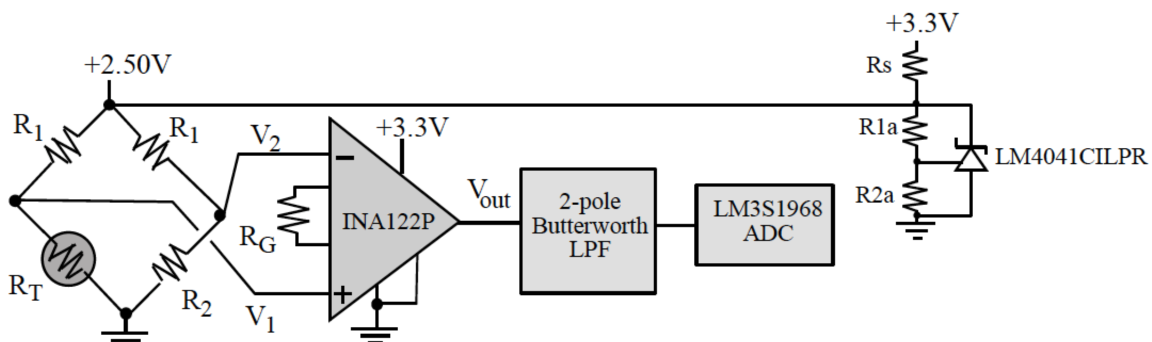
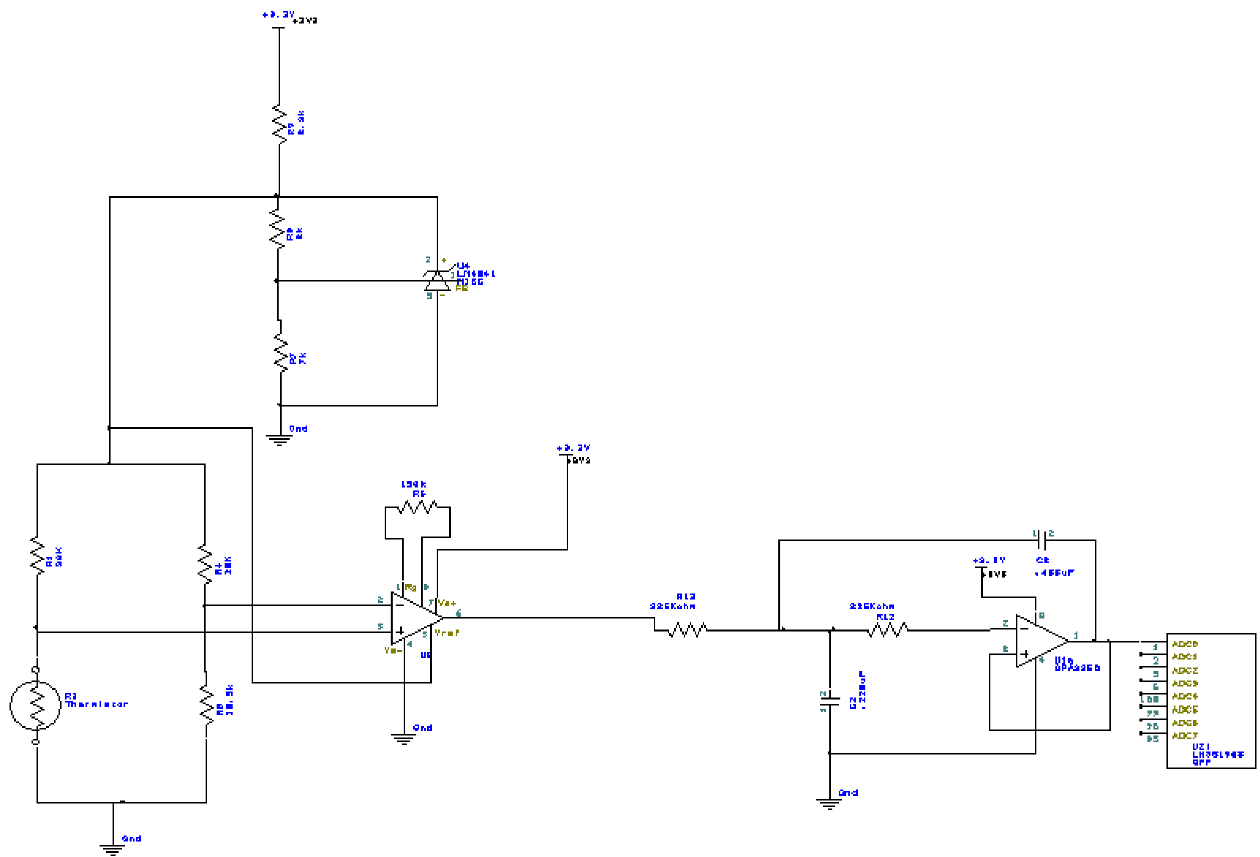


Figure 9.3. Possible thermistor interface (easy to construct, but more expensive).

2.2 Circuit diagram of the thermistor interface



3.0 Software Design

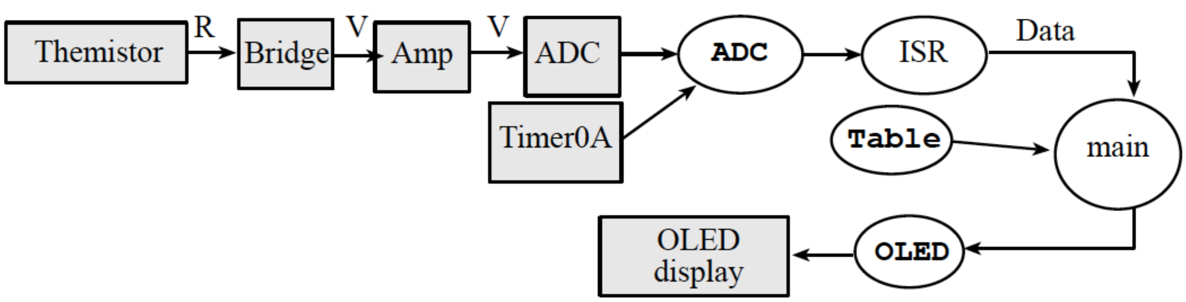


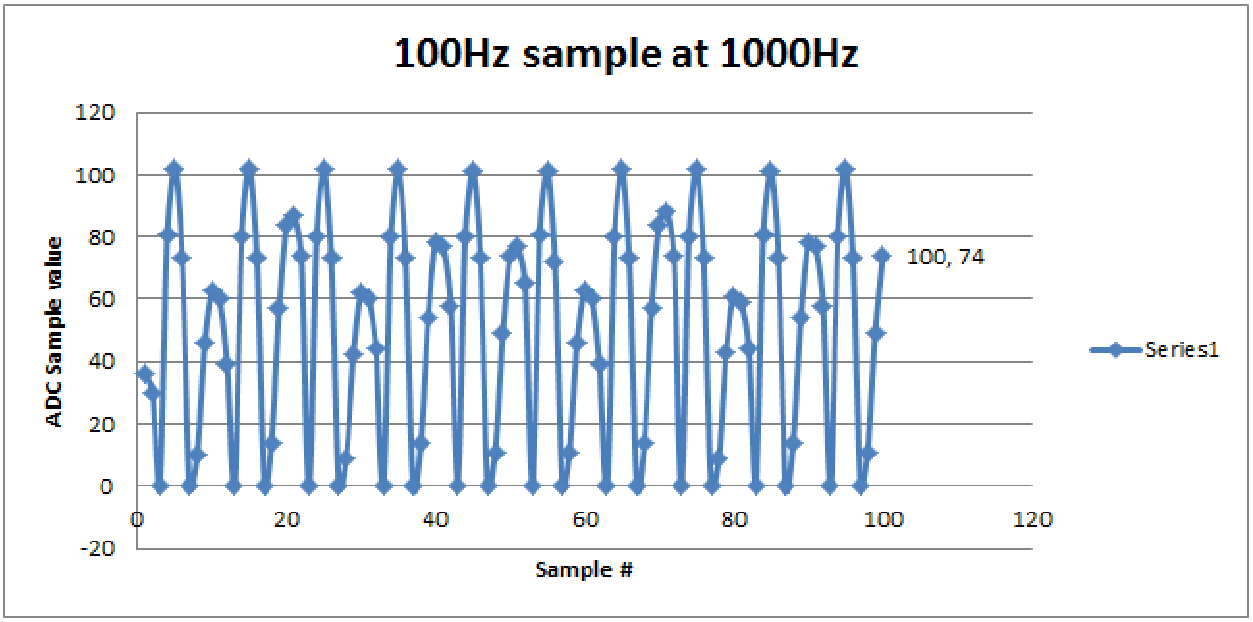
Figure 9.1. Data-flow graph of the data acquisition system (R means resistance, V means voltage).

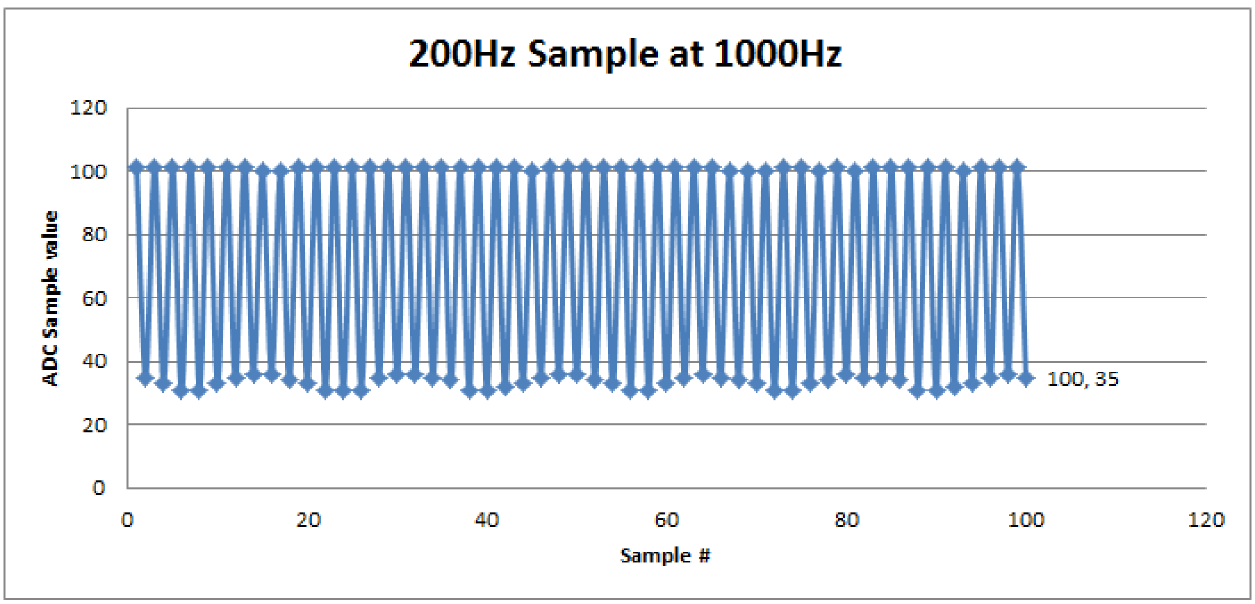
Below code are at the end of this report:

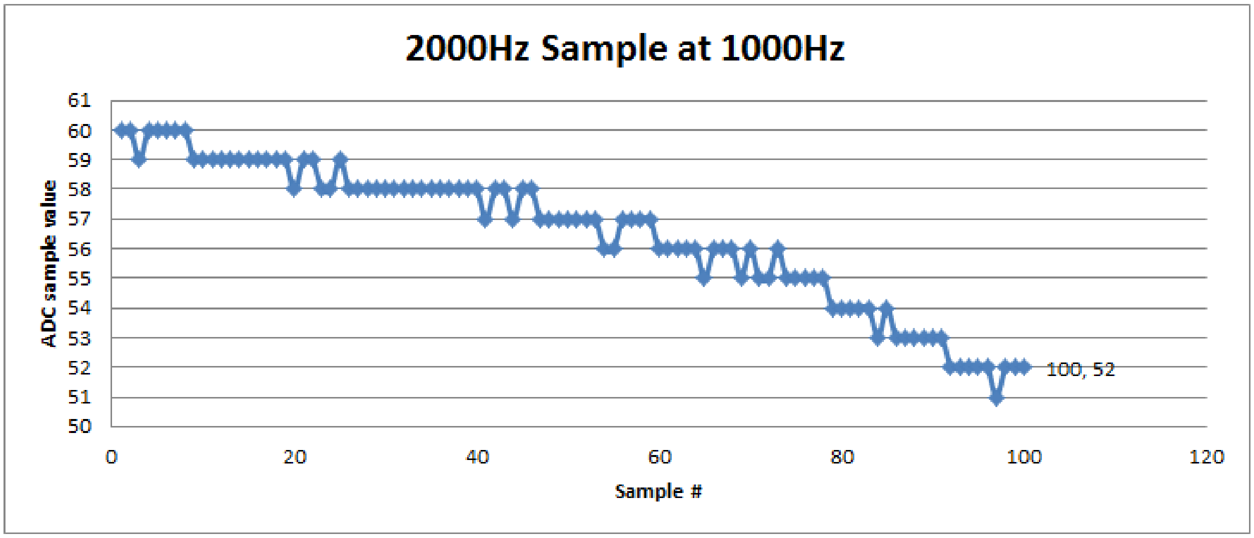
1) Calibration data (procedure 5 and the calib.h file)
2) Main program used to measure temperature (ADC is embedded in calibration)

4.0 Measurement Data

4.1 Sketch three waveforms (procedure 1)







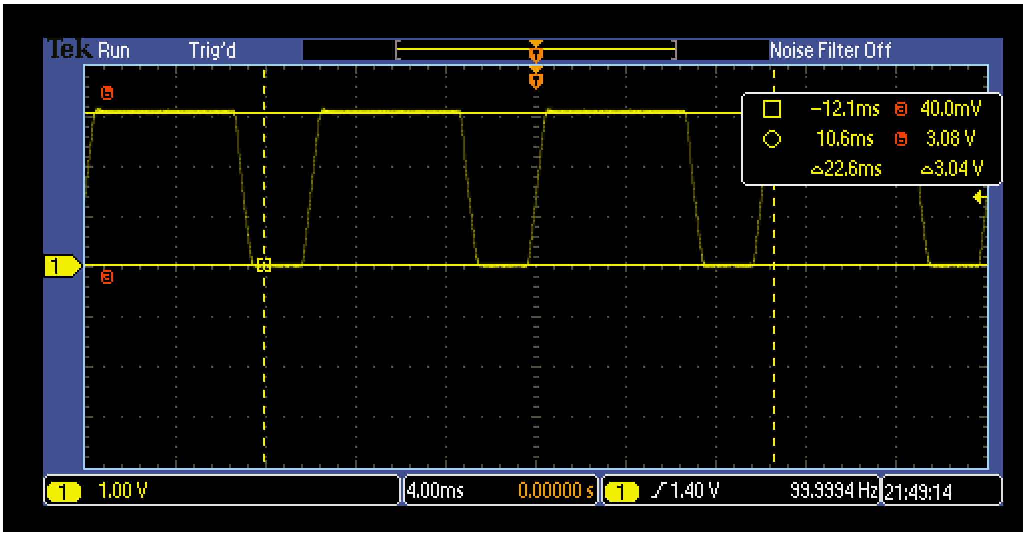
4.2 Static circuit performance (procedure 2,4)

ADC	RT (Ω)	Test R (Ω)	Vs (V)	Vs measured (V)	ADC displayed
1023	∞	disconnected	3.25	3.25	1023
1015	1206	1200	2.978	2.96	1013
792	997	1000	2.325	2.29	793
107	495	500	0.314	0.3	110
76	470	470	0.225	0.21	72
0	0	shorted	0	0.00047	0

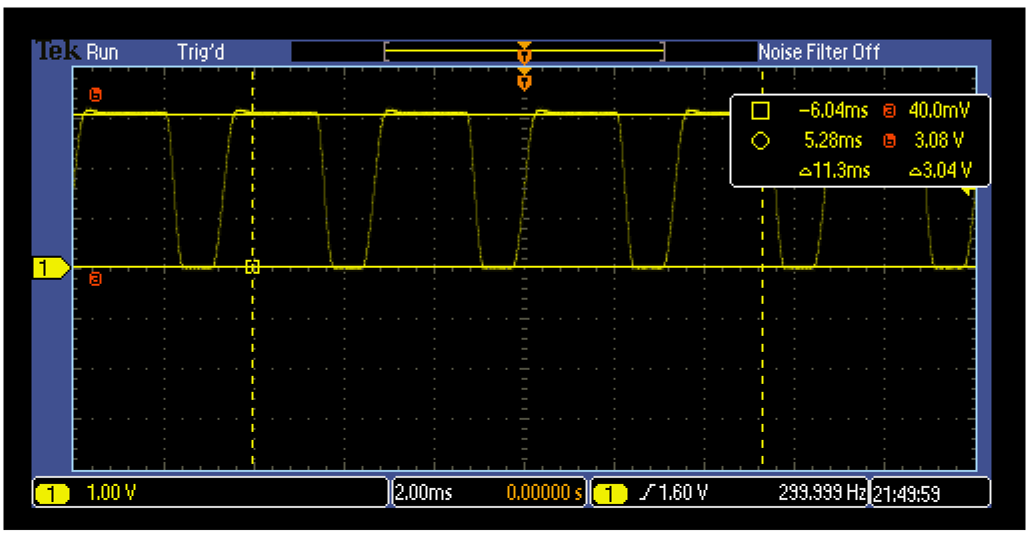
4.3 Dynamic circuit performance (procedure 3)

-10 measurement was captured, will display five below:

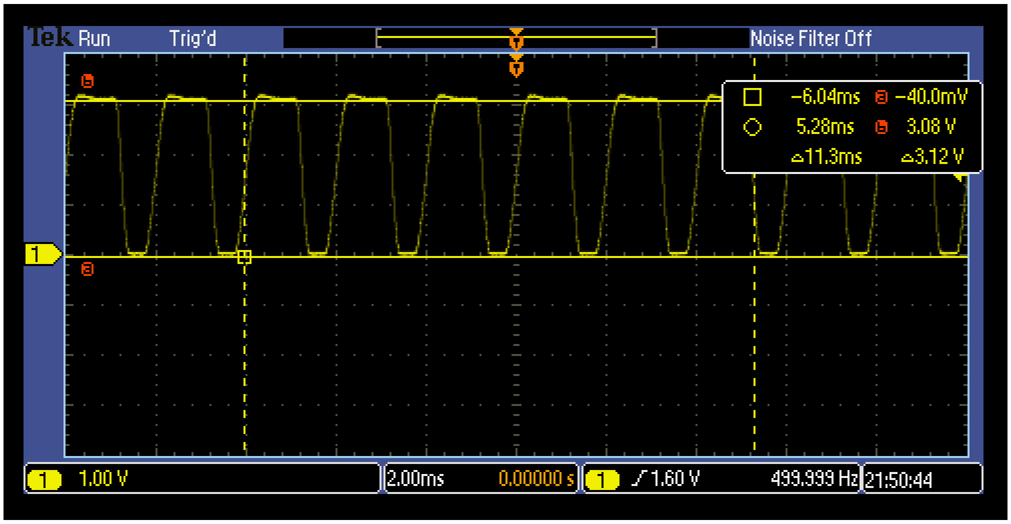
Supply with amplitude 1.250 V ,+.700 V and 100Hz



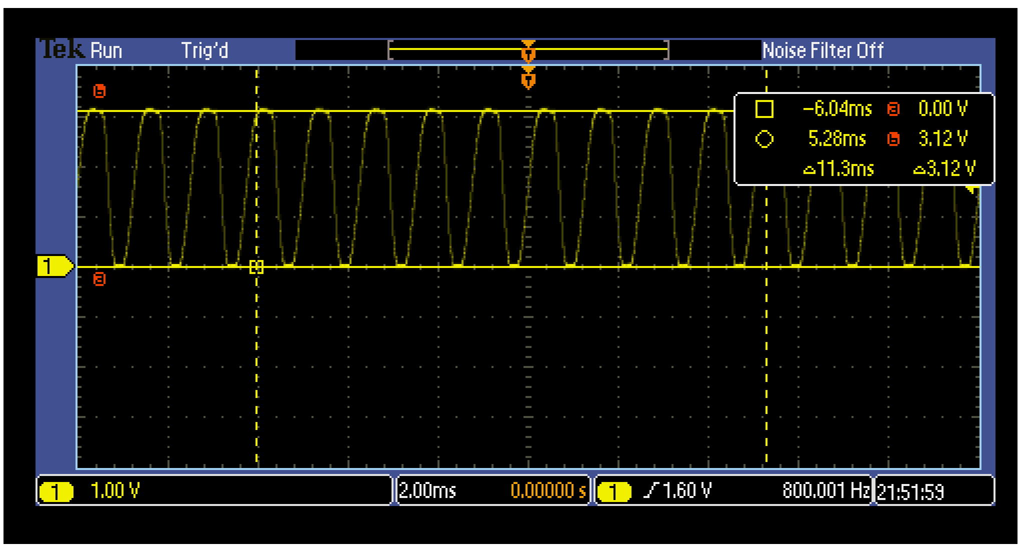
Supply with amplitude 1.250 V ,+.700 V and 300Hz



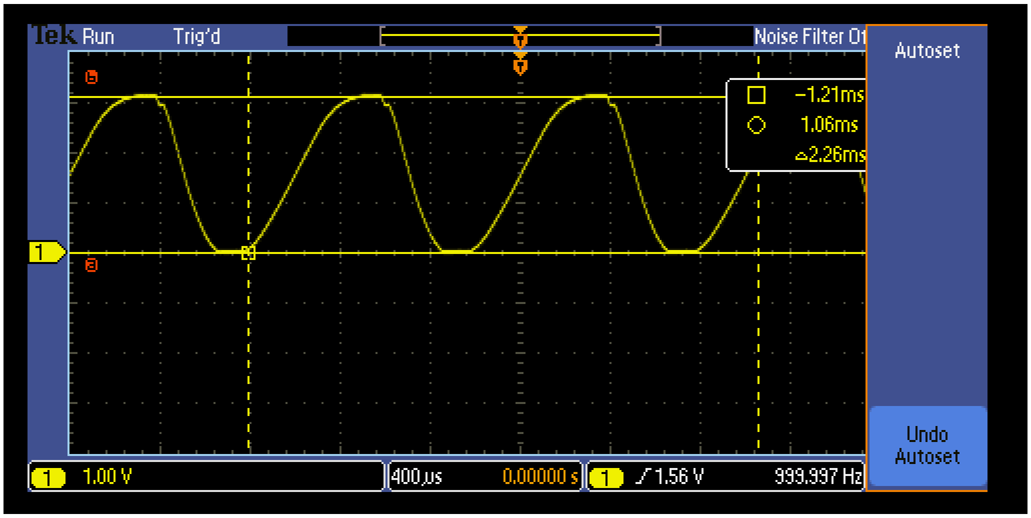
Supply with amplitude 1.250 V ,+.700 V and 500Hz



Supply with amplitude 1.250 V ,+.700 V and 800Hz



Supply with amplitude 1.250 V ,+.700 V and 800Hz



'Average gain for 10 measurement is about 1.248

4.4 Accuracy (procedure 6)

True Temperature (̊ C)	Temperature Displayed (̊ C)	ΔT (̊ C)
24	23.52	0.48
25	24.69	0.31
26	25.83	0.17
33	33.06	0.06

Average Accuracy = 0.255̊ C

4.5 Reproducibility (procedure 7)

Temperature (̊C)
24.52
24.43
24.46
24.53
24.59
24.65
24.72
24.66
24.69
24.62

Standard Deviation = 0.098

5.0 Analysis and Discussion (give short answers to these questions)

1) What is the Nyquist theorem and how does it apply to this lab?

Nyquist theorem is a sampling theorem. It stated that the sampling rate must be at least two time the processed signal frequency. In this lab the temperature signal is between 0 to 10 Hz, hence we need a sampling rate of 20Hz or higher.

2) Explain the difference between resolution and accuracy?

Accuracy depend on resolution in addition to calibration drift.

.

3) Derive an equation to relate reproducibility and precision of the thermometer.

$$\begin{aligned}\sigma &= \sqrt{\mathrm{E}[(X-\mu)^2]} \\ &= \sqrt{\mathrm{E}[X^2] + \mathrm{E}[(-2\mu X)] + \mathrm{E}[\mu^2]} = \sqrt{\mathrm{E}[X^2] - 2\mu\,\mathrm{E}[X] + \mu^2} \\ &= \sqrt{\mathrm{E}[X^2] - 2\mu^2 + \mu^2} = \sqrt{\mathrm{E}[X^2] - \mu^2} \\ &= \sqrt{\mathrm{E}[X^2] - (\mathrm{E}[X])^2}.\end{aligned}$$

where σ is standard deviation, E is the expected value, X is a random variable, and μ is the average. We calculated the standard deviation to be roughly 0.098. Therefore each temperature deviated within 0.1°C making the system reproduce numbers that are roughly the same.

4) What is the purpose of the LPF?

The purpose of the low pass filter is to pass low frequencies and attenuate the higher frequencies to prevent anti-aliasing.

5) If the R versus T curve of the thermistor is so nonlinear, why does the voltage versus temperature curve look so linear?

Voltage conversion is being linearized and calibrated manually while the low level non-linear output must be condition properly for a linear correlation.

6) There are four methods (a,b,c,d) listed in the 4) Software Conversion section of methods and constraints. For one of the methods you did not implement, give reasons why your method is better, and give reasons why this alternative method would have been better.

We did not implement a linear method because the temperature versus thermistor resistance is not linear rather it is an exponential function. By linearly interpolating values between the minimum temperature and the maximum temperature would be incorrect and therefore result in large errors. However, if we took 1024 samples and linearly interpolated the values in between each sample, then we get much more accurate results in smaller error.

6.0 CODE

```
//Filename: Calib.h
//Author: Duc Tran, Brandon Wong
//Initial Creation Date: November 10, 2013
//Description:
//Lab Number: W 2-3:30
//TA : Omar & Mahesh
//Date of last revision : November 18, 2013
//Hardware Configuration : Lab 9.SCH
```

```
/***************Find_ADCindex***************
 Get Index of ADC value in ADC conversion array.
 Input: unsigned short raw ADC value
 Output: unsigned short index of ADC conversion array

 */
unsigned short Find_ADCindex(unsigned short ADC);


/***************Interpolate***************
 Convert raw ADC value to a temperature representation.
 Input: Index of ADC value in conversion array ,unsigned short raw ADC value
 Output: unsigned short temperature value
 */
unsigned short Interpolate(unsigned short index, unsigned short ADC);




// ADCPrintResults.c
// Runs on LM3S811
// Use a setup similar to ADCT0ATrigger.c to gather ADC samples
// into a buffer.  When the buffer is full, print them to the
// UART separated by TABs.  The purpose of this is to verify
// that the measured data from the ADC matches what is expected
// for a known input.  If the input signal is a sine wave, then
// the measured data should look like a sine wave.
// Daniel Valvano
// October 12, 2011
// Last Edited By: Duc Tran, Brandon Wong
// November 18,2013

/* This example accompanies the book
   "Embedded Systems: Real Time Interfacing to the Arm Cortex M3",
   ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2011

 Copyright 2011 by Jonathan W. Valvano, valvano@mail.utexas.edu
    You may use, edit, run or distribute this file
    as long as the above copyright notice remains
 THIS SOFTWARE IS PROVIDED "AS IS".  NO WARRANTIES, WHETHER EXPRESS, IMPLIED
 OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
```

```
// Sine wave from signal generator connected to ADC1
//
// This program periodically samples ADC channel 1 and stores the
// result to a buffer.  After the buffer is full, it stops
// triggering ADC conversions and outputs the results to the UART
// separated by TABs.

#include "hw_types.h"
#include "sysctl.h"
#include "UART.h"
#include "FIFO.h"
#include "Calib.h"
#include "Fixed.h"
#include "Output.h"
#include "rit128x96x4.h"

#define NVIC_EN0_INT17          0x00020000  // Interrupt 17 enable
#define NVIC_EN0_R              (*((volatile unsigned long *)0xE000E100))  // IRQ 0 to 31 Set Enable
Register
#define NVIC_PRI4_R             (*((volatile unsigned long *)0xE000E410))  // IRQ 16 to 19 Priority
Register
#define TIMER0_CFG_R            (*((volatile unsigned long *)0x40030000))
#define TIMER0_TAMR_R           (*((volatile unsigned long *)0x40030004))
#define TIMER0_CTL_R            (*((volatile unsigned long *)0x4003000C))
#define TIMER0_IMR_R            (*((volatile unsigned long *)0x40030018))
#define TIMER0_TAILR_R          (*((volatile unsigned long *)0x40030028))
#define TIMER0_TAPR_R           (*((volatile unsigned long *)0x40030038))
#define TIMER_CFG_16_BIT        0x00000004  // 16-bit timer configuration,
                                // function is controlled by bits
                                // 1:0 of GPTMTAMR and GPTMTBMR
#define TIMER_TAMR_TAMR_PERIOD  0x00000002  // Periodic Timer mode
#define TIMER_CTL_TAOTE         0x00000020  // GPTM TimerA Output Trigger
                                // Enable
#define TIMER_CTL_TAEN          0x00000001  // GPTM TimerA Enable
```

```
#define TIMER_IMR_TATOIM        0x00000001  // GPTM TimerA Time-Out Interrupt
                                            // Mask
#define TIMER_TAILR_TAILRL_M    0x0000FFFF  // GPTM TimerA Interval Load
                                            // Register Low
#define ADC_ACTSS_R             (*((volatile unsigned long *)0x40038000))
#define ADC0_RIS_R              (*((volatile unsigned long *)0x40038004))
#define ADC0_IM_R               (*((volatile unsigned long *)0x40038008))
#define ADC0_ISC_R              (*((volatile unsigned long *)0x4003800C))
#define ADC0_EMUX_R             (*((volatile unsigned long *)0x40038014))
#define ADC0_SSPRI_R            (*((volatile unsigned long *)0x40038020))
#define ADC0_PSSI_R             (*((volatile unsigned long *)0x40038028))
#define ADC0_SSMUX3_R           (*((volatile unsigned long *)0x400380A0))
#define ADC0_SSCTL3_R           (*((volatile unsigned long *)0x400380A4))
#define ADC0_SSFIFO3_R          (*((volatile unsigned long *)0x400380A8))
#define ADC_ACTSS_ASEN3         0x00000008  // ADC SS3 Enable
#define ADC_RIS_INR3            0x00000008  // SS3 Raw Interrupt Status
#define ADC_IM_MASK3            0x00000008  // SS3 Interrupt Mask
#define ADC_ISC_IN3             0x00000008  // SS3 Interrupt Status and Clear
#define ADC_EMUX_EM3_M          0x0000F000  // SS3 Trigger Select mask
#define ADC_EMUX_EM3_TIMER      0x00005000  // Timer
#define ADC_SSPRI_SS3_4TH       0x00003000  // fourth priority
#define ADC_SSPRI_SS2_3RD       0x00000200  // third priority
#define ADC_SSPRI_SS1_2ND       0x00000010  // second priority
#define ADC_SSPRI_SS0_1ST       0x00000000  // first priority
#define ADC_PSSI_SS3            0x00000008  // SS3 Initiate
#define ADC_SSMUX3_MUX0_M       0x00000003  // 1st Sample Input Select mask
#define ADC_SSMUX3_MUX0_S       0           // 1st Sample Input Select lshift
#define ADC_SSCTL3_TS0          0x00000008  // 1st Sample Temp Sensor Select
#define ADC_SSCTL3_IE0          0x00000004  // 1st Sample Interrupt Enable
#define ADC_SSCTL3_END0         0x00000002  // 1st Sample is End of Sequence
#define ADC_SSCTL3_D0           0x00000001  // 1st Sample Diff Input Select
#define ADC_SSFIFO3_DATA_M      0x000003FF  // Conversion Result Data mask
#define GPIO_PORTG_DATA_R       (*((volatile unsigned long *)0x400263FC))
#define GPIO_PORTG_DIR_R        (*((volatile unsigned long *)0x40026400))
#define GPIO_PORTG_DEN_R        (*((volatile unsigned long *)0x4002651C))
#define SYSCTL_RCGC0_R          (*((volatile unsigned long *)0x400FE100))
#define SYSCTL_RCGC1_R          (*((volatile unsigned long *)0x400FE104))
#define SYSCTL_RCGC2_R          (*((volatile unsigned long *)0x400FE108))
#define SYSCTL_RCGC0_ADC        0x00010000  // ADC0 Clock Gating Control
#define SYSCTL_RCGC0_ADCSPD_M   0x00000300  // ADC Sample Speed mask
#define SYSCTL_RCGC0_ADCSPD500K 0x00000200  // 500K samples/second
#define SYSCTL_RCGC1_TIMER0     0x00010000  // timer 0 Clock Gating Control
#define SYSCTL_RCGC2_GPIOG      0x00000040  // port G Clock Gating Control
```

```
#define MAXBUFFERSIZE          100         // maximum number of samples
#define SAMPLEFREQ            1000       // sampling frequency (min. 763 Hz)
#define CLOCKFREQ             50000000    // PLL clock frequency


#define ADC2CHSUCCESS 1 // represents success reading from FIFO
#define ADC2CHFAILED  0 // represents failed reading from FIFO

// FIFO macros
AddIndexFifo(Ch1,MAXBUFFERSIZE,unsigned short,ADC2CHSUCCESS,ADC2CHFAILED)
// creates Ch1Fifo_Init() Ch1Fifo_Get() and Ch1Fifo_Put()

void DisableInterrupts(void); // Disable interrupts
void EnableInterrupts(void);  // Enable interrupts
long StartCrit1ical (void);    // previous I bit, disable interrupts
void EndCritical(long sr);    // restore I bit to previous value
void WaitForInterrupt(void);  // low power mode

// There are many choices to make when using the ADC, and many
// different combinations of settings will all do basically the
// same thing.  For simplicity, this function makes some choices
// for you.  When calling this function, be sure that it does
// not conflict with any other software that may be running on
// the microcontroller.  Particularly, ADC sample sequencer 3
// is used here because it only takes one sample, and only one
// sample is absolutely needed.  Sample sequencer 3 generates a
// raw interrupt when the conversion is complete, and it is then
// promoted to an ADC controller interrupt.  Hardware Timer0A
// triggers the ADC conversion at the programmed interval, and
// software handles the interrupt to process the measurement
// when it is complete.
//
// A simpler approach would be to use software to trigger the
// ADC conversion, wait for it to complete, and then process the
// measurement.
//
// This initialization function sets up the ADC according to the
// following parameters.  Any parameters not explicitly listed
// below are not modified:
// Timer0A: enabled
// Mode: 16-bit, down counting
// One-shot or periodic: periodic
// Prescale value: programmable using variable 'prescale' [0:255]
```

```
// Interval value: programmable using variable 'period' [0:65535]
// Sample time is busPeriod*(prescale+1)*(period+1)
// Max sample rate: <=500,000 samples/second
// Sequencer 0 priority: 1st (highest)
// Sequencer 1 priority: 2nd
// Sequencer 2 priority: 3rd
// Sequencer 3 priority: 4th (lowest)
// SS3 triggering event: Timer0A
// SS3 1st sample source: programmable using variable 'channelNum' [0:3]
// SS3 interrupts: enabled and promoted to controller
void ADC_InitTimer0ATriggerSeq3(unsigned char channelNum, unsigned char prescale, unsigned short
period){
  volatile unsigned long delay;
  // channelNum must be 0-3 (inclusive) corresponding to ADC0 through ADC3
  if(channelNum > 3){
    return;                       // invalid input, do nothing
  }
  DisableInterrupts();
  // **** general initialization ****
  SYSCTL_RCGC0_R |= SYSCTL_RCGC0_ADC;      // activate ADC
  SYSCTL_RCGC0_R &= ~SYSCTL_RCGC0_ADCSPD_M; // clear ADC sample speed field
  SYSCTL_RCGC0_R += SYSCTL_RCGC0_ADCSPD500K;// configure for 500K ADC max sample
rate
  SYSCTL_RCGC1_R |= SYSCTL_RCGC1_TIMER0;   // activate timer0
  delay = SYSCTL_RCGC1_R;                 // allow time to finish activating
  TIMER0_CTL_R &= ~TIMER_CTL_TAEN;        // disable timer0A during setup
  TIMER0_CTL_R |= TIMER_CTL_TAOTE;        // enable timer0A trigger to ADC
  TIMER0_CFG_R = TIMER_CFG_16_BIT;        // configure for 16-bit timer mode
  // **** timer0A initialization ****
  TIMER0_TAMR_R = TIMER_TAMR_TAMR_PERIOD;  // configure for periodic mode
  TIMER0_TAPR_R = prescale;               // prescale value for trigger
  TIMER0_TAILR_R = period;                // start value for trigger
  TIMER0_IMR_R &= ~TIMER_IMR_TATOIM;      // disable timeout (rollover) interrupt
  TIMER0_CTL_R |= TIMER_CTL_TAEN;         // enable timer0A 16-b, periodic, no interrupts
  // **** ADC initialization ****
                              // sequencer 0 is highest priority (default setting)
                              // sequencer 1 is second-highest priority (default setting)
                              // sequencer 2 is third-highest priority (default setting)
                              // sequencer 3 is lowest priority (default setting)
  ADC0_SSPRI_R =
(ADC_SSPRI_SS0_1ST|ADC_SSPRI_SS1_2ND|ADC_SSPRI_SS2_3RD|ADC_SSPRI_SS3_4TH);
  ADC_ACTSS_R &= ~ADC_ACTSS_ASEN3;        // disable sample sequencer 3
  ADC0_EMUX_R &= ~ADC_EMUX_EM3_M;         // clear SS3 trigger select field
```

```
  ADC0_EMUX_R += ADC_EMUX_EM3_TIMER;        // configure for timer trigger event
  ADC0_SSMUX3_R &= ~ADC_SSMUX3_MUX0_M;      // clear SS3 1st sample input select field
                                            // configure for 'channelNum' as first sample input
  ADC0_SSMUX3_R += (channelNum<<ADC_SSMUX3_MUX0_S);
  ADC0_SSCTL3_R = (0                        // settings for 1st sample:
                & ~ADC_SSCTL3_TS0        // read pin specified by ADC0_SSMUX3_R (default setting)
                | ADC_SSCTL3_IE0         // raw interrupt asserted here
                | ADC_SSCTL3_END0        // sample is end of sequence (default setting, hardwired)
                & ~ADC_SSCTL3_D0);       // differential mode not used (default setting)
  ADC0_IM_R |= ADC_IM_MASK3;                // enable SS3 interrupts
  ADC_ACTSS_R |= ADC_ACTSS_ASEN3;           // enable sample sequencer 3
  // **** interrupt initialization ****
                                            // ADC3=priority 2
  NVIC_PRI4_R = (NVIC_PRI4_R&0xFFFF00FF)|0x00004000; // bits 13-15
  NVIC_EN0_R |= NVIC_EN0_INT17;             // enable interrupt 17 in NVIC
  EnableInterrupts();
}
volatile unsigned short indexBuff = 0;
volatile unsigned long ADCbuffer[MAXBUFFERSIZE];
void ADC3_Handler(void){
  ADC0_ISC_R = ADC_ISC_IN3;                 // acknowledge ADC sequence 3 completion
  GPIO_PORTG_DATA_R ^= 0x04;                // toggle LED
  Ch1Fifo_Put(ADC0_SSFIFO3_R&ADC_SSFIFO3_DATA_M);
//  ADCbuffer[index] = ADC0_SSFIFO3_R&ADC_SSFIFO3_DATA_M;
  indexBuff = indexBuff + 1;
  if(indexBuff == MAXBUFFERSIZE){
    ADC_ACTSS_R &= ~ADC_ACTSS_ASEN3;    // disable sample sequencer 3
    TIMER0_CTL_R &= ~TIMER_CTL_TAEN;    // disable timer0A
  }
}

int main(void){
  unsigned short i,j;
        int test = 1;
        unsigned short ADCValue,average;
        unsigned short TempValue;
        char TempString;
        char Welcome ;
        sprintf(&Welcome,"Current Temperature");
  //
  // Set the system clock to run at 50 MHz from the main oscillator.
  //
  SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL |
```

```
            SYSCTL_XTAL_8MHZ | SYSCTL_OSC_MAIN);
  SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOG; // activate port G
        Output_Init();
        Output_Color(15);
  UART_Init();   // initialize the UART as a console for text I/O
  GPIO_PORTG_DIR_R |= 0x04;             // make PG2 out (PG2 built-in LED)
  GPIO_PORTG_DEN_R |= 0x04;             // enable digital I/O on PG2
  GPIO_PORTG_DATA_R &= ~0x04;           // turn off LED
  UART_OutString("\r\n\nADC Test Program");

  while(1){
   UART_OutString("\r\n\nSample frequency (Hz): ");
   UART_OutUDec(SAMPLEFREQ);
   UART_OutString("\r\nNumber of samples: ");
   UART_OutUDec(MAXBUFFERSIZE);
   UART_OutString("\r\nPress ENTER to begin sampling.\r\n");
   while(UART_InChar() != 13){};       // wait for ENTER key
   indexBuff = 0;                       // reset counter
                           // ADC channel 1, SAMPLEFREQ Hz sampling
   ADC_InitTimer0ATriggerSeq3(0, 0, (CLOCKFREQ/SAMPLEFREQ)-1);
                          for(i=1;i<20;i=i+1){
                                  UART_OutChar(13);
                                  UART_OutChar(10);
                                  average = 0;
                                  Ch1Fifo_Get(&ADCValue);
                                  UART_OutUDec(ADCValue);
                                  TempValue = Interpolate(Find_ADCindex(ADCValue),ADCValue);
                                  Fixed_uDecOut2s(TempValue,&TempString);
                                  RIT128x96x4StringDraw(&Welcome,10,20,12);
                                  RIT128x96x4StringDraw(&TempString,20,60,12);
                                  UART_OutString(&TempString);
                          }

  }
}
```