EE445L – Lab 9: Temperature Data Acquisition System

Michael Park and Jack Zhao

04/18/16

1.0 OBJECTIVE

The objective of this lab is to design, build and test a digital thermometer that can measure the range of 10 to 40 °C using an ADC converter on the TM4C123. Educationally, we are learning how to create a temperature acquisition circuit using a Whitestone Bridge circuit, instrumentation amplifier, and a butterworth low-pass filter as well as how to choose a sampling frequency (Nyquist Theorem, Valvano Postulate) and a cutoff frequency for the filter.

2.0 HARDWARE DESIGN

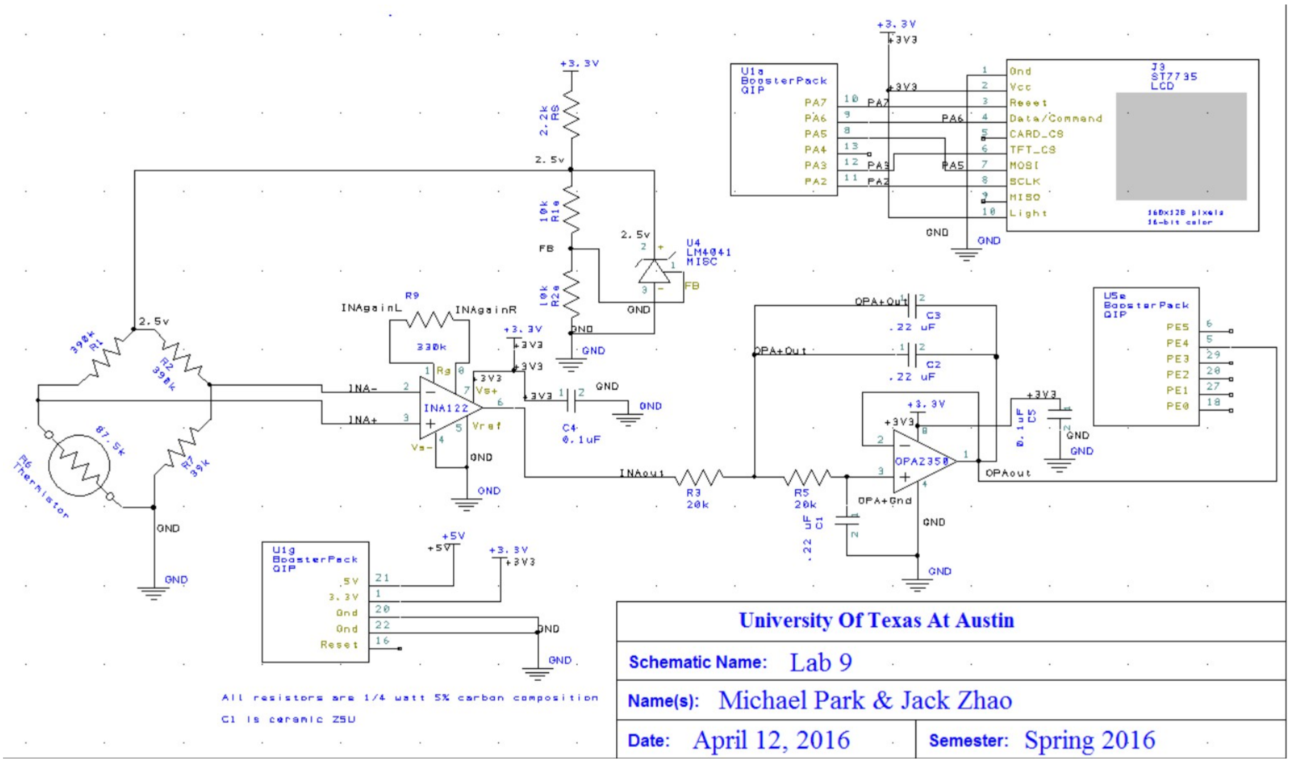

Figure 1: Circuit diagram of the thermistor interface

3.0 SOFTWARE DESIGN

1) Calibration data (procedure 5 and the calib.h file)

```
// ******* Calib.h **************
// Michael Park, Jack Zhao
// Date Created: 04/09/2016
// Includes prototypes for functions Temp Calibration
// Lab Number: 16340
// TA: Mahesh Srinivasan
// Last Revised: 04/15/2016
uint16_t const ADCTable[53]={0,184,226,269,313,358,404,452,501,550,601,

     654,707,762,818,876,935,995,1057,1120,1184,
     1251,1318,1388,1459,1531,1605,1681,1759,1838,1919,

     2002,2086,2173,2261,2351,2443,2537,2632,2730,2829,

     2931,3034,3140,3247,3356,3467,3580,3695,3812,3931,4052,4096};

uint16_t const TempTable[53]={4000,4000,3940,3880,3820,3760,3700,3640,3600,3550,3510,
     3510,3500,3490,3480,3470,3450,3400,3350,3300,3250,
     3200,3000,2800,2680,2560,2450,2380,2370,2369,2350,
     2330,2300,2280,2270,2260,2200,1840,1780,1720,1660,
     1600,1540,1480,1420,1360,1300,1240,1180,1120,1060,1000,1000};
```

2) Low level ADC interface (ADC.c and ADC.h files)

```
// ******* ADCSWTrigger.c **************
// Michael Park, Jack Zhao
// Date Created: 04/09/2016
// Includes function definition for ADC
// Lab Number: 16340
// TA: Mahesh Srinivasan
// Last Revised: 04/15/2016

// ADCSWTrigger.c
// Runs on LM4F120/TM4C123
// Provide functions that initialize ADC0 SS3 to be triggered by
// software and trigger a conversion, wait for it to finish,
// and return the result.
// Daniel Valvano
// May 2, 2015

/* This example accompanies the book
   "Embedded Systems: Real Time Interfacing to Arm Cortex M Microcontrollers",
   ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2015

 Copyright 2015 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
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS
SOFTWARE.
```

For more information about my classes, my research, and my books, see
http://users.ece.utexas.edu/~valvano/
*/

```c
#include <stdint.h>
#include "ADCSWTrigger.h"
#include "../inc/tm4c123gh6pm.h"

// 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, ADC0 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, but it is not
// promoted to a controller interrupt.  Software triggers the
// ADC0 conversion and waits for the conversion to finish.  If
// somewhat precise periodic measurements are required, the
// software trigger can occur in a periodic interrupt.  This
// approach has the advantage of being simple.  However, it does
// not guarantee real-time.
//
// A better approach would be to use a hardware timer to trigger
// the ADC0 conversion independently from software and generate
// an interrupt when the conversion is finished.  Then, the
// software can transfer the conversion result to memory and
// process it after all measurements are complete.

// This initialization function sets up the ADC according to the
// following parameters.  Any parameters not explicitly listed
// below are not modified:
// Max sample rate: <=125,000 samples/second
// Sequencer 0 priority: 1st (highest)
// Sequencer 1 priority: 2nd
// Sequencer 2 priority: 3rd
// Sequencer 3 priority: 4th (lowest)
// SS3 triggering event: software trigger
// SS3 1st sample source: Ain9 (PE4)
// SS3 interrupts: enabled but not promoted to controller

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

uint32_t ADCValue;
```

```
void ADC0_InitSWTriggerSeq3_Ch9(uint32_t period){
	volatile uint32_t delay;
	                       // 1) activate clock for Port E
  SYSCTL_RCGCGPIO_R |= 0x10;
  while((SYSCTL_PRGPIO_R&0x10) != 0x10){};

  GPIO_PORTE_DIR_R &= ~0x10;      // 2) make PE4 input
  GPIO_PORTE_AFSEL_R |= 0x10;     // 3) enable alternate function on PE4
  GPIO_PORTE_DEN_R &= ~0x10;      // 4) disable digital I/O on PE4
  GPIO_PORTE_AMSEL_R |= 0x10;     // 5) enable analog functionality on PE4
  SYSCTL_RCGCADC_R |= 0x0001;   // 7) activate ADC0
  while((SYSCTL_PRADC_R&0x0001) != 0x0001){};    // good code, but not yet implemented in
simulator

  ADC0_PC_R &= ~0xF;              // 7) clear max sample rate field
  ADC0_PC_R |= 0x1;               //    configure for 125K samples/sec
  ADC0_SSPRI_R = 0x0123;          // 8) Sequencer 3 is highest priority


	SYSCTL_RCGCTIMER_R |= 0x01;                              // activate timer0
	delay = SYSCTL_RCGCGPIO_R;
	TIMER0_CTL_R = 0x00000000;                     //disable timer0A for setup
	TIMER0_CTL_R |= 0x00000020;                    //enable timer0A trigger to ADC
	TIMER0_CFG_R = 0;                                                    //configure
for 32-bit timer mode
	TIMER0_TAMR_R = 0x00000002;                    //configures for periodic mode
	TIMER0_TAPR_R = 0;                                            //prescale value for
trigger
	TIMER0_TAILR_R = period-1;                     //start value for trigger
	TIMER0_IMR_R = 0x00000000;                     //disable all interrupts
	TIMER0_CTL_R |= 0x00000001;                    //enable timer0A 32-b periodic


  ADC0_ACTSS_R &= ~0x0008;        // 9) disable sample sequencer 3
	ADC0_EMUX_R = (ADC0_EMUX_R&0xFFFF0FFF) + 0x5000; //timer trigger
  //ADC0_EMUX_R &= ~0xF000;         // 10) seq3 is software trigger
  ADC0_SSMUX3_R &= ~0x000F;       // 11) clear SS3 field
  ADC0_SSMUX3_R += 9;             //    set channel
  ADC0_SSCTL3_R = 0x0006;         // 12) no TS0 D0, yes IE0 END0
  ADC0_IM_R &= ~0x0008;           // 13) disable SS3 interrupts
  ADC0_ACTSS_R |= 0x0008;         // 14) enable sample sequencer 3


	NVIC_PRI4_R = (NVIC_PRI4_R&0xFFFF00FF) | 0x00004000; //priority 2
	NVIC_EN0_R = 1<<17;                                           //enable interrupt
17 in NVIC
	EnableInterrupts();
}

//------------ADC0_InSeq3------------
// Busy-wait Analog to digital conversion
// Input: none
```

```
// Output: 12-bit result of ADC conversion
uint32_t ADC0_InSeq3(void){  uint32_t result;
  ADC0_PSSI_R = 0x0008;            // 1) initiate SS3
  while((ADC0_RIS_R&0x08)==0){};   // 2) wait for conversion done
    // if you have an A0-A3 revision number, you need to add an 8 usec wait here
  result = ADC0_SSFIFO3_R&0xFFF;   // 3) read result
  ADC0_ISC_R = 0x0008;             // 4) acknowledge completion
  return result;
}

//------------ADC0Seq3_Handler------------
// Interrupts on adc change
// Input: none
// Output: none
void ADC0Seq3_Handler(void){
//  ADC0_PSSI_R = 0x0008;            // 1) initiate SS3
//  while((ADC0_RIS_R&0x08)==0){};   // 2) wait for conversion done
// if you have an A0-A3 revision number, you need to add an 8 usec wait here
	ADC0_ISC_R = 0x08;             // 4) acknowledge completion
  ADCValue = ADC0_SSFIFO3_R&0xFFF;   // 3) read result
  ADC0_ISC_R = 0x0008;             // 4) acknowledge completion
}

//------------returnADC------------
// return adc value
// Input: none
// Output: ADC
uint32_t returnADC(void){
	return ADCValue;
}

// ******* ADCSWTrigger.h **************
// Michael Park, Jack Zhao
// Date Created: 04/09/2016
// Includes prototypes for functions in ADCSWTrigger.c
// Lab Number: 16340
// TA: Mahesh Srinivasan
// Last Revised: 04/15/2016


// ADCSWTrigger.h
// Runs on LM4F120/TM4C123
// Provide functions that initialize ADC0 SS3 to be triggered by
// software and trigger a conversion, wait for it to finish,
// and return the result.
// Daniel Valvano
// May 2, 2015

/* This example accompanies the book
   "Embedded Systems: Real Time Interfacing to Arm Cortex M Microcontrollers",
   ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2015
```

```
// 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, ADC0 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, but it is not
// promoted to a controller interrupt.  Software triggers the
// ADC0 conversion and waits for the conversion to finish.  If
// somewhat precise periodic measurements are required, the
// software trigger can occur in a periodic interrupt.  This
// approach has the advantage of being simple.  However, it does
// not guarantee real-time.
//
// A better approach would be to use a hardware timer to trigger
// the ADC0 conversion independently from software and generate
// an interrupt when the conversion is finished.  Then, the
// software can transfer the conversion result to memory and
// process it after all measurements are complete.

// This initialization function sets up the ADC according to the
// following parameters.  Any parameters not explicitly listed
// below are not modified:
// Max sample rate: <=125,000 samples/second
// Sequencer 0 priority: 1st (highest)
// Sequencer 1 priority: 2nd
// Sequencer 2 priority: 3rd
// Sequencer 3 priority: 4th (lowest)
// SS3 triggering event: software trigger
// SS3 1st sample source: Ain9 (PE4)
// SS3 interrupts: enabled but not promoted to controller
void ADC0_InitSWTriggerSeq3_Ch9(uint32_t period);
```

```
//------------ADC0_InSeq3------------
// Busy-wait Analog to digital conversion
// Input: none
// Output: 12-bit result of ADC conversion
uint32_t ADC0_InSeq3(void);

void ADC0Seq3_Handler(void);

uint32_t returnADC(void);
```
3) Main program used to measure temperature
```
// ******* Main.c **************
// Michael Park, Jack Zhao
// Date Created: 04/9/2016
// Main program functions
// Lab Number: 16340
// TA: Mahesh Srinivasan
// Last Revised: 04/15/2016
// main.c
// Runs on LM4F120/TM4C123
// UART runs at 115,200 baud rate
// Daniel Valvano
// May 3, 2015

/* This example accompanies the books
  "Embedded Systems: Introduction to ARM Cortex M Microcontrollers",
  ISBN: 978-1469998749, Jonathan Valvano, copyright (c) 2015

"Embedded Systems: Real Time Interfacing to ARM Cortex M Microcontrollers",
   ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2015

 Copyright 2015 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
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS
SOFTWARE.
 VALVANO SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL,
INCIDENTAL,
 OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 For more information about my classes, my research, and my books, see
 http://users.ece.utexas.edu/~valvano/
 */


#include <stdio.h>
#include <stdint.h> // C99 variable types
#include "ADCSWTrigger.h"
#include "uart.h"
#include "PLL.h"
#include "ST7735.h"
```

```
#include "Timer1.h"


/* Assuming 80MHz Bus
1khz : 80000
100hz(Valvano Postulate) : 800000
500hz(Nyquist Theorom) :  160000
2khz(Aliased) :  40000 */
#define ADCfreq 80000

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

void periodicADC(void);
int32_t ADCTempConvert(uint32_t ADC);
int32_t Interpolate(int lower, int higher, int32_t ADC);
void lcdFixed(int32_t temp);
volatile uint32_t counter;
volatile int32_t data[100];

int main(void){
	counter=0;
  PLL_Init(Bus80MHz);   // 80 MHz
  ST7735_InitR(INITR_REDTAB);                                          // Init PORTA and LCD
initializations
  ST7735_FillScreen(ST7735_WHITE);
  ST7735_FillRect(0, 0, 128, 50, ST7735_BLACK);
	ST7735_PlotClear(0, 4000);

  UART_Init();              // initialize UART device
  ADC0_InitSWTriggerSeq3_Ch9(4000000);//4000000

	Timer1_Init(*periodicADC,800 );//4000000
EnableInterrupts();

	while(1){};
}

void periodicADC(void){
	uint32_t ADCval;
	uint32_t TEMPval;
	//ADCval=returnADC();
	ADCval=ADC0_InSeq3();
	//TEMPval=ADCTempConvert(ADCval);
	//UART_OutUDec(TEMPval);
	//prelab

	/*
	//data[counter]=TEMPval;
```

```
		data[counter]=ADC0_InSeq3();
	if(counter>=100){
		for(int k=0;k<20;k++){
			UART_OutString("\n\r");
		}
		UART_OutString("\n\r--------------------Starting--------------------");
		for(int j=0; j<100; j++){
			UART_OutString("\n\r");
			UART_OutUDec(j);
			UART_OutString("): ------ ADC data =");
			UART_OutUDec(data[counter]);
		}
		while(1){};
	}
	counter++;

	*/
	//end prelab

	TEMPval=ADCTempConvert(ADCval);
	lcdFixed(TEMPval);
	ST7735_SetCursor(0,2);
	ST7735_OutString("ADC=");
	ST7735_OutUDec(ADCval);
	ST7735_PlotNextErase();
	ST7735_PlotPoint(TEMPval);
	ST7735_PlotNext();
	ST7735_PlotNextErase();
}
int32_t ADCTempConvert(uint32_t ADC){
	int point=0;
	while(ADCTable[point]<ADC){
		point++;
	}
	if(ADCTable[point]==ADC){
		return TempTable[point];
	}
	return Interpolate(point-1,point,ADC);
}

int32_t Interpolate(int lower, int higher, int32_t ADC){
	double tempLower=TempTable[lower];
	double tempHigher=TempTable[higher];
	double adcLower=ADCTable[lower];
	double adcHigher=ADCTable[higher];
	double currentADC=ADC;

	double result= tempLower+ (tempHigher-tempLower) * ((currentADC-adcLower)/
(adcHigher-adcLower));
	return (int32_t) result;
}
void lcdFixed(int32_t temp){
```

```
        char out[6];
        ST7735_SetCursor(0,0);
        ST7735_OutString("Thermometer");
        ST7735_SetCursor(0,1);
        ST7735_OutString("T=");

        if(temp>99999){
                ST7735_DrawString(0,0,"***.**",ST7735_WHITE);
        }
        else{
                int before=temp/100;
                int after=temp%100;
                sprintf(out,"%d.%d",before,after);
                ST7735_OutString(out);
                ST7735_OutString(" C");


        }
}
```

4.0 MEASUREMENT DATA

1) Sketch three waveforms (procedure 1)

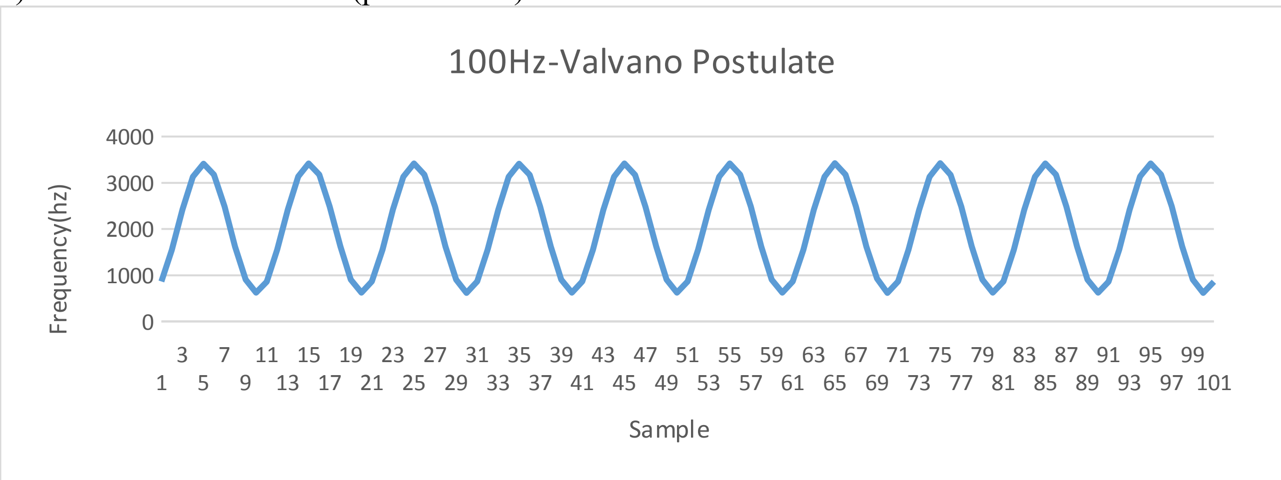

Figure 2: Valvano Postulate Sine Wave

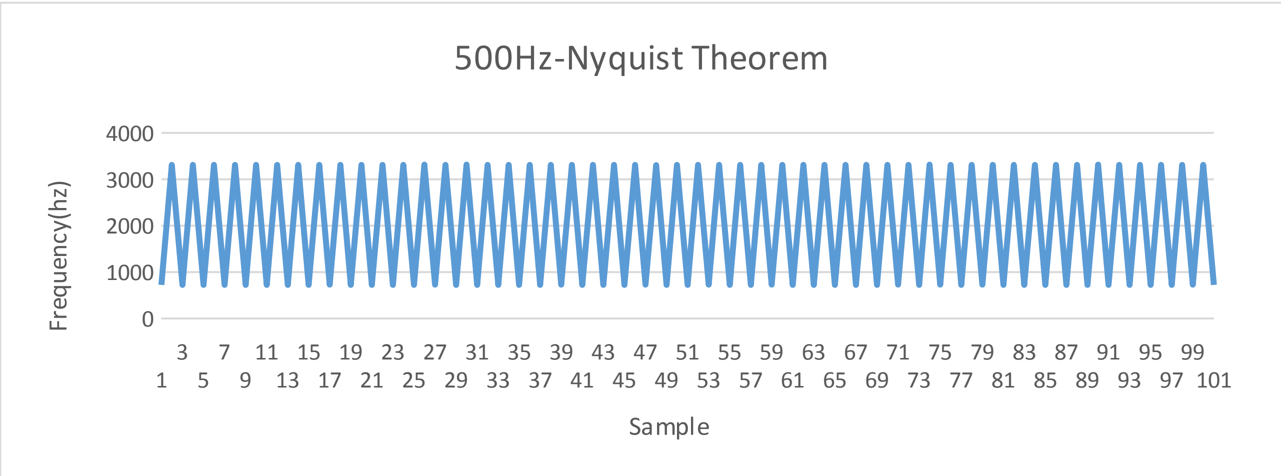

Figure 3: Nyquist Theorem Sine Wave

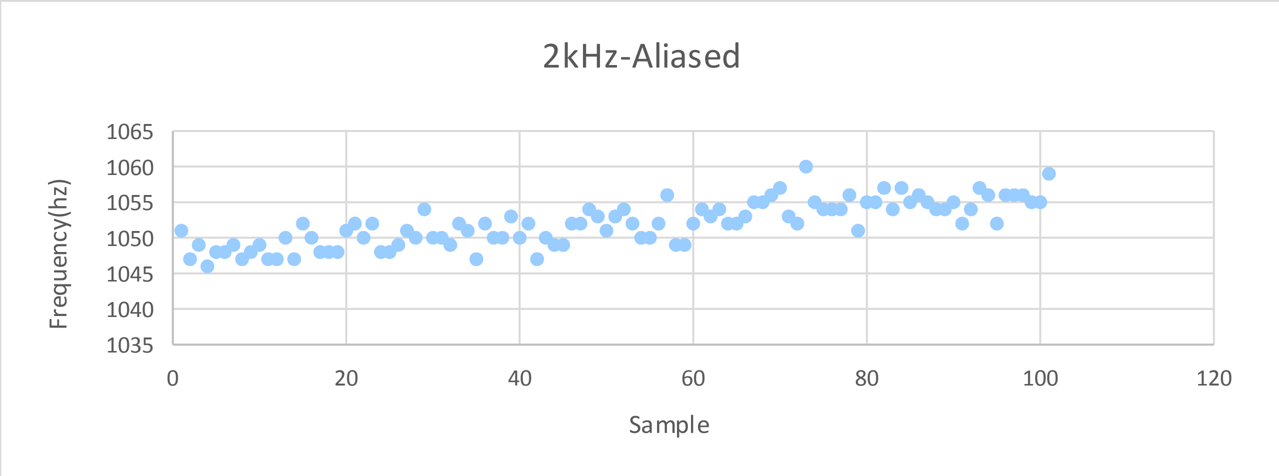

Figure 4: Aliased Sine Wave

The Valvano postulate specifies to sample at 10x the frequency and the Nyquist Theorem states to sample at 2x the frequency. The Aliased is sampling slower than the frequency. From the graphs, we can see the Valvano postulate graph display a much more defined graph and the Nyquist theorem just capture key points on the frequency. The Aliased graph does not completely capture the sine wave.

2) Static circuit performance (procedure 2)

181k	3.04V
127k	2.08V
100k	1.52V
54k	640 mV
disconnected	3.24 V
shorted	500mV

3) Dynamic circuit performance (procedure 3)

input sinewave amp	100mV-560mV	
output sinewave	561mV-3.14V	
un amplified	amplified	Gain
400mV	1.60V	4
472mV	1.72V	3.64
456mV	1.80V	3.94
580mV	2.60V	4.48
640mV	2.96V	4.625
720mV	3.04V	4.22
728mV	3.08V	4.23
750mV	3.10V	4.13
580mV	3.16V	5.44
520mV	3.16V	6.07

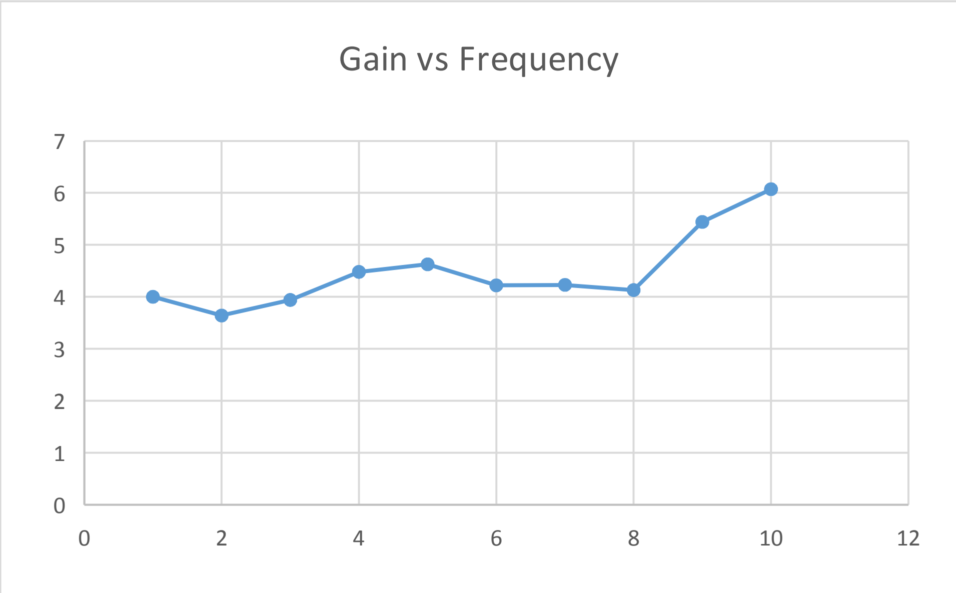

Figure 5: Gain vs Frequency

4) Accuracy (procedure 6)

Measured	Fluke
23.46	23.45
23.46	23.47
23.45	23.41
23.48	23.44
23.47	23.44

avg error: .026

5) Reproducibility (procedure 7)

23.46
23.46
23.45
23.48
23.47
23.48
23.48
23.46
23.45
23.47

Standard Deviation: .01174
S: .00014

5.0 ANALYSIS AND DISCUSSION

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

Nyquist theorem states that the sampling frequency should be at least twice the frequency of the signal to be sampled. In this lab, the frequency component of the measured signal was between 0 – 10 Hz. Therefore the sampling frequency of our system had to be at least 20 Hz.

2) Explain the difference between resolution and accuracy?

Resolution is the smallest change in the input parameter that can be reliably detected by the measurement. Accuracy ((Actual – Ideal) / Ideal) is a measure of how close our instrument measures the desired parameter referred to the NIST.

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

Reproducibility is a parameter that specifies whether the instrument has equal outputs give identical inputs over some period of time. This parameter can be expressed as the full range of output results given a fixed input. Thus, the precision of the thermometer can be expressed as the following equation.

Precision = full range of output results at fixed input / resolution of the thermometer.

4) What is the purpose of the LPF?

The purpose of the LPF is to remove high frequency noise to improve signal to noise ratio. In this lab, our frequency of interest was 0 – 10Hz (low frequency). Thus using a butterworth LPF(Anti-ailiasing filter) restricted the bandwith of the signal, and as a result, increased the signal to noise ratio (SNR).

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

The R versus T curve of the thermistor is nonlinear becuase the transducer(thermistor) is nonlinear. However, the bridge circuit used for the thermistor is also nonlinear as shown by the equation (2.5*Rt)/(R1+Rt). Therefore, because both the transducer and the circuit are nonlinear, the final outcome (voltage vs temperature curve) becomes much closer to being linear.

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 used a small table lookup (≈50 entries) with linear interpolation in between instead of a large table lookup (one entry for each ADC value). Small table lookup takes up less memory space than the large table lookup. On the other hand, large table lookup is more accurate because it does not use linear interpolation.