Ronald Macmaster and Parth Adhia

EID: Rpm953 and Pda375

11/11/2016

EE 445L – Lab 9:

Temperature Data Acquisition System

1.0: Objectives:

We will further study the concepts of ADC conversion and the Nyquist Theorem. Specifically, we plan to develop a temperature measurement system that uses a thermistor. We plan to analyze different performance measurements of the system including precision, resolution, and accuracy. To realize a working system, we will also have to understand some general concepts of analog circuit design. Our circuit will require the use of an operational amplifier and instrumentation amplifier. We need to calibrate these analog circuits in a way that will allow for the most accurate temperature measurements in the overall system. That means we also must gain a deep understanding of how these integrated circuits work and the parameters associated with them.

2.0: Hardware Design:

Our system will interact with a 50k Thermistor to measure the external environment temperature. The thermistor will be interfaced through a resistor bridge. Two bridge voltage references will be input into an instrumentation amplifier, INA122PA. Then, the instrumentation amplifier output is sent to the low pass filter. The low pass filter is implemented with an OPA2350 operational amplifier, and it has a critical frequency of ~100Hz. We set the cutoff frequency to the sampling frequency of our acquisition system. The primary purpose of the low pass filter is to eliminate signals that may alias the temperature data. **The Thermistor interface can be found on the next page.**

Thermistor Interface

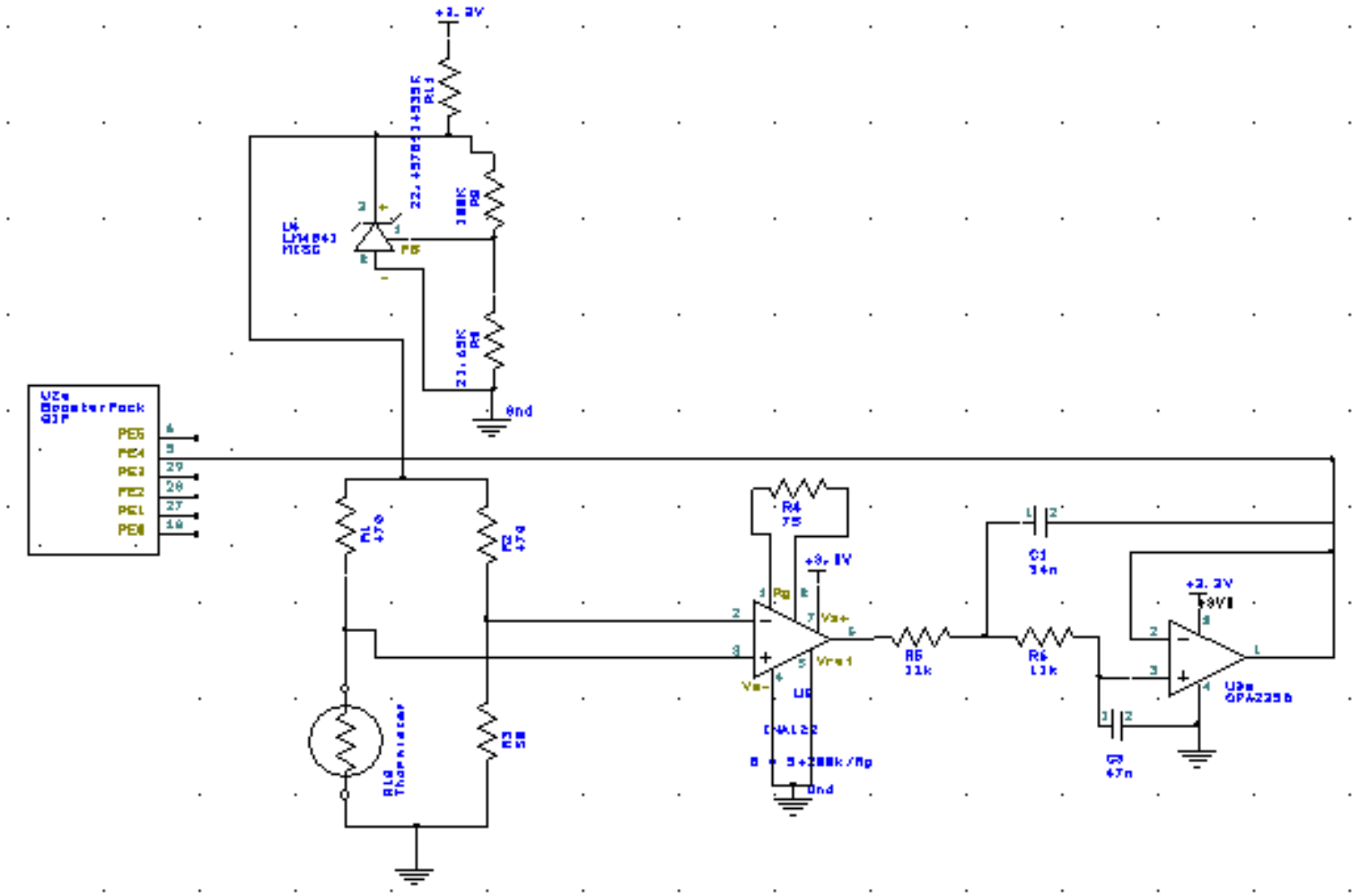

Figure 2.1: *Hardware Schematic for our Temperature Data Acquisition System. The Thermistor will increase its resistance when its temperature drops, and it will decrease its resistance when its temperature rises. The INA instrumentation amp serves to eliminate common-mode noise, and the low pass filter eliminates aliasing signals into the ADC.*

3.0: Software Design:

All of our required and additional functionality for our temperature system can be implemented using four basic modules — Switch input, LCD output, and ADC input. Submitted along with this report are the interface and driver files (.h and .c) for our software modules (ADC, LCD, Plot, Timer, and FIFO).

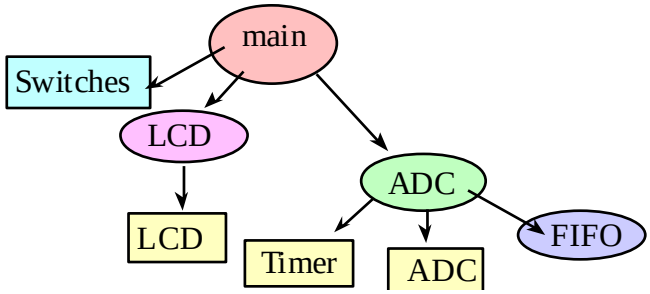

Figure 3.1: *Call graph for our Temperature Data Acquisition System. A main driver program manages hardware through secondary software modules. The ADC is triggered through hardware output compare, and the samples are buffered in a software FIFO.*

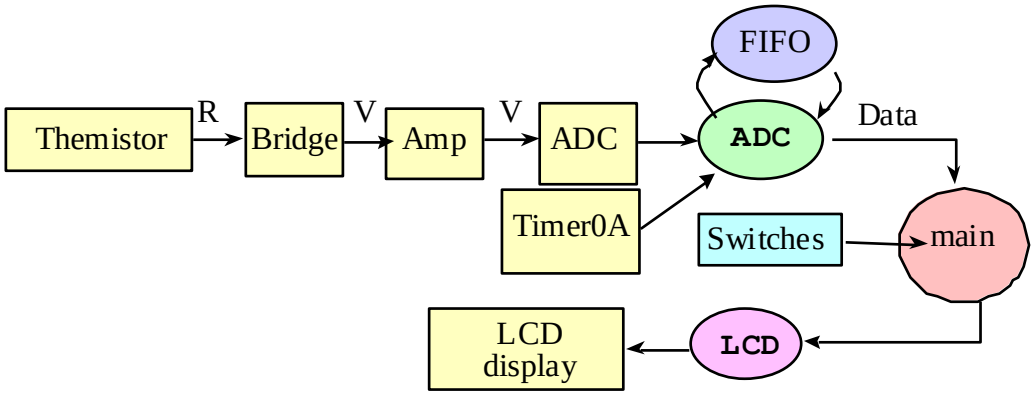

Figure 3.2: *Data flow graph for our Temperature Data Acquisition System. The thermistor voltage is filtered of common node noise and aliasing signals through the bridge and amplifier circuits. The sample is read from the ADC and output to the LCD screen as a monitoring plot. The plot updates constantly.*

```c
/** calibration data
 * R1 = 470k    R2 = 470k
 * R3 = 50k
 *
 * calibration points (T, Rt)
 * First point:  ( , )
 * Second point: ( , )
 */
#include <stdint.h>

const uint16_t MAP_SIZE = 53;
uint16_t const ADCdata[53]={0,105,153,203,253,305,357,411,465,521,579,
      637,696,757,819,882,947,1013,1080,1148,1218,
      1289,1362,1436,1511,1588,1666,1746,1827,1909,1993,
      2079,2166,2254,2344,2436,2529,2623,2719,2817,2916,
      3016,3118,3221,3326,3432,3540,3649,3760,3872,3985,4095,4096};

uint16_t const Tdata[53]={4000,4000,3940,3880,3820,3760,3700,3640,3580,3520,3460,
      3400,3340,3280,3220,3160,3100,3040,2980,2920,2860,
      2800,2740,2680,2620,2560,2500,2440,2380,2320,2260,
      2200,2140,2080,2020,1960,1900,1840,1780,1720,1660,
      1600,1540,1480,1420,1360,1300,1240,1180,1120,1060,1000,1000};
```

Figure 3.3: *Calibration data for our Thermistor. The two arrays act as a lookup table for our thermistor interpolation.*

ADCHWTrigger.c | main.c | startup.s

```c
// Sequencer 3 priority: 4th (lowest)
// SS3 triggering event: Timer0A
// SS3 1st sample source: programmable using variable 'channelNum' [0:11]
// SS3 interrupts: enabled and promoted to controller
void ADC0_InitHWTrigger(uint32_t period){
  PortE_Init();
  Timer0A_Init(period, 0);
  volatile uint32_t delay;
  SYSCTL_RCGCADC_R |= 0x01;     // activate ADC0
  delay = SYSCTL_RCGCTIMER_R;   // allow time to finish activating
  delay = SYSCTL_RCGCTIMER_R;   // allow time to finish activating

  ADC0_PC_R = 0x01;         // configure for 125K samples/sec
  ADC0_SSPRI_R = 0x3210;    // sequencer 0 is highest, sequencer 3 is lowest
  ADC0_ACTSS_R &= ~0x08;    // disable sample sequencer 3
  ADC0_EMUX_R = (ADC0_EMUX_R&0xFFFF0FFF)+0x5000; // timer trigger event
  ADC0_SSMUX3_R = 9;
  ADC0_SSCTL3_R = 0x06;          // set flag and end
  ADC0_IM_R |= 0x08;             // enable SS3 interrupts
  ADC0_ACTSS_R |= 0x08;          // enable sample sequencer 3
  ADC0_SAC_R = ADC_SAC_AVG_64X;   // enable Hardware Averaging
  NVIC_PRI4_R = (NVIC_PRI4_R&0xFFFF00FF)|0x00004000; //priority 2
  NVIC_EN0_R = 1<<17;              // enable interrupt 17 in NVIC
  Timer0A_Start();
  EnableInterrupts();
}
void ADC0Seq3_Handler(void){
  ADC0_ISC_R = 0x08;          // acknowledge ADC sequence 3 completion
  ADCvalue = ADC0_SSFIFO3_R;  // 12-bit result
  FIFO_Write(ADCvalue); // write to data queue.
}
```

Figure 3.4: *Low-level ADC Interface. ADC is sampled on-call from a hardware timer trigger. Once the sample conversion process has completed, the ADC requests an interrupt. The handler places the sample into a software queue.*

```
51 int main(void){
52   uint32_t data;
53   PLL_Init(Bus80MHz);   // 80 MHz
54   LCD_Init(); // screen debugging
55   UART_Init(); // initialize UART device
56   ADC0_InitHWTrigger(TIMER_100Hz);
57
58   Plot_Init("Temperature plot", 4096, 0);
59   for(int idx = 0; ; idx++){
60     data = ADC0_In();
61     UART_OutString("\n\rADC data ="); UART_OutUDec(data);
62     Plot_PrintData(data);
63     Plot_PlotData(data);
64   }
65 }
66
67
```

Figure 3.5: *Main driver for our Data Acquisition System. Data is continuously sampled and output to the LCD Plot.*

4.0: Measurement Data:

Measurements were acquired as they were asked for in the lab manual.

4.1: Nyquist Theorem Waveforms

Plot the three waveforms (100Hz, 500Hz, 2000Hz) by connecting the data points with a straight line. Describe the concepts of Nyquist Theorem, Valvano Postulate, and aliasing using this data.

Analysis: The Nyquist theorem generally states that you can only accurately represent signals that are bandlimited by ½ the original sampling frequency, f_s. Valvano furthered this notion with his postulate, you must sample at 10x the maximum frequency for the signal to look like the original when plotted on a graph. If the sample is bandlimited by a frequency greater than ½ f_s, the frequency components above ½ f_s will become indistinguishable from the lower frequency components. These components will distort the original signal.

These theoretical concepts are reflected well in our data plots. The 100Hz signal is easily represented and supported by the Valvano postulate. The 500Hz signal is somewhat reflected. We can deduce that the original sample is 500Hz, but it looks nothing like the original signal. The 2000Hz signal is completely aliased, and we have no way of deducing the original signal from the sample data.

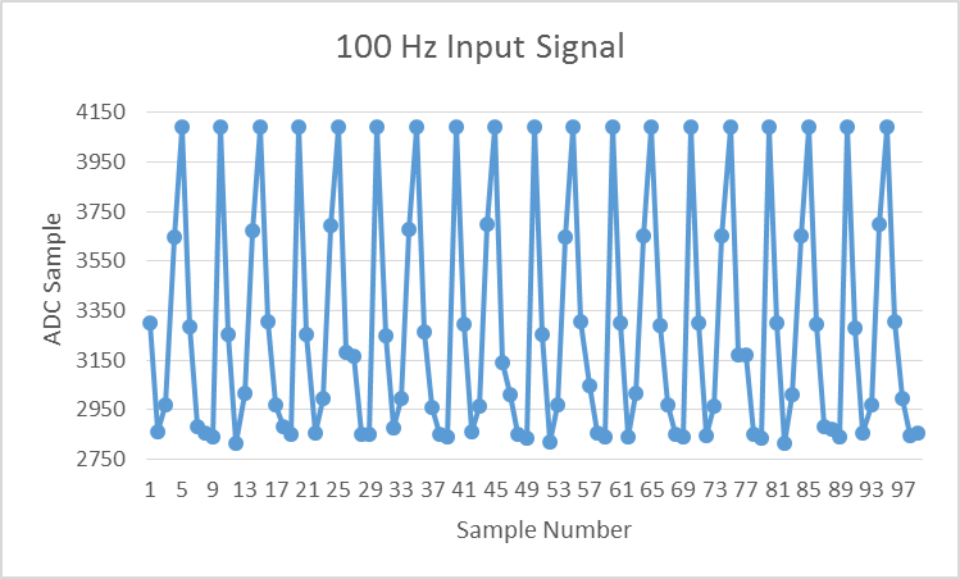

Figure 4.1: 100 Samples at 100Hz. Fs = 1000Hz. (Valvano Postulate).

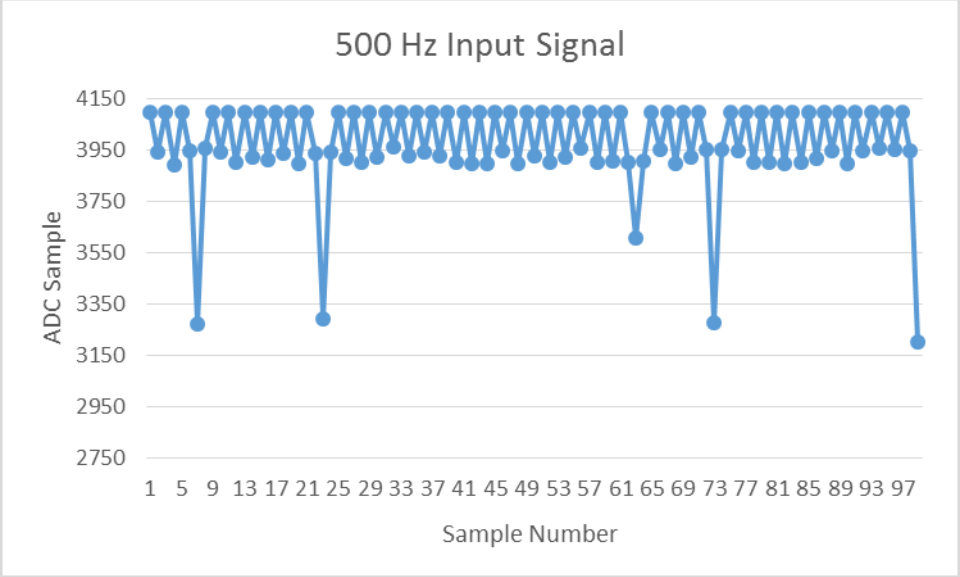

Figure 4.2: 100 Samples at 500Hz. Fs = 1000Hz. (Nyquist Theorem).

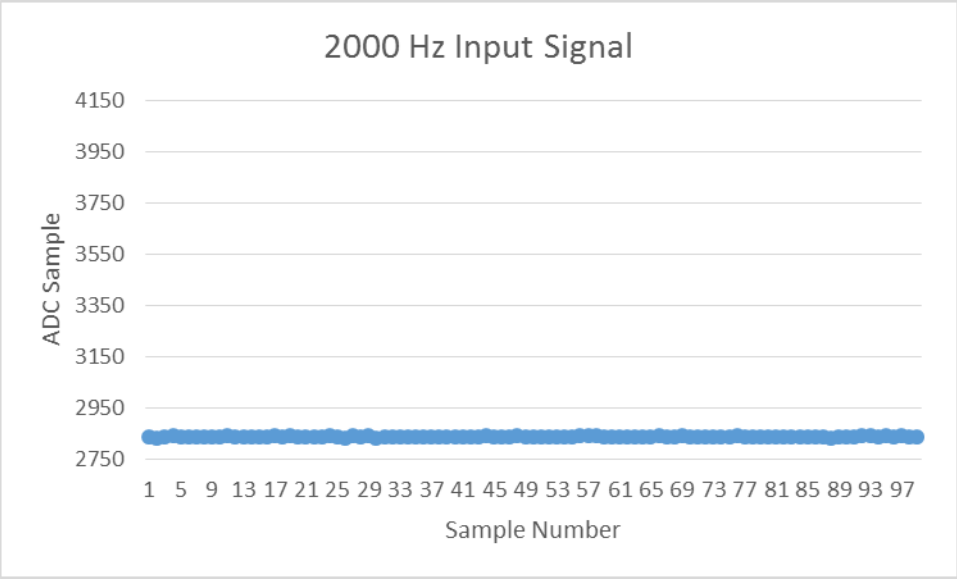

Figure 4.3: 100 Samples at 2000Hz. Fs = 1000Hz. (Aliasing).

4.2: Static Circuit Performance

Record the voltage values at strategic places in your analog circuit. What voltage output do you get when the thermistor is disconnected? What voltage output do you get when the thermistor wires are shorted?

When Rt is	Therm voltage into INA [V]	Voltage out of INA [V]
Rt min : 62 k	0.01	3.24
Rt2 : 62.978 K	0.4	0.441
Rt3 74.3 K	0.459	0.751
Rt4: 81.6 K	0.496	0.943
Rt5: 82.7 k	0.501	0.967
Rt6: 98.6 K	0.575	1.362
Rt max: 99.6 K	0.58	1.385
open	2.378	3.269
short	0	0.0011

Figure 4.4: Static analysis on the circuit at Thermistor extremes and within range.

4.3: Dynamic Circuit Test

Record the sine-wave amplitudes of the input and output voltages. Calculate the gain at each frequency. Plot the gain versus frequency response of your circuit.

Amplitude of input sine wave: 2.5V

Frequency [Hz]	Amplitude into LPF	Amplitude out of LPF	Gain (out / in)
1	2.48	2.3	0.927419355
5	2.48	2.24	0.903225806
10	2.4	2.24	0.933333333
50	2.8	1.92	0.685714286
100	2.8	1.52	0.542857143
200	2.8	0.8	0.285714286
500	2.8	0.24	0.085714286
1000	2.8	0.16	0.057142857
5000	2.8	0.16	0.057142857
10000	2.8	0.08	0.028571429
20000	2.8	0.08	0.028571429

Figure 4.5: Frequency Response of the circuit's low pass filter.

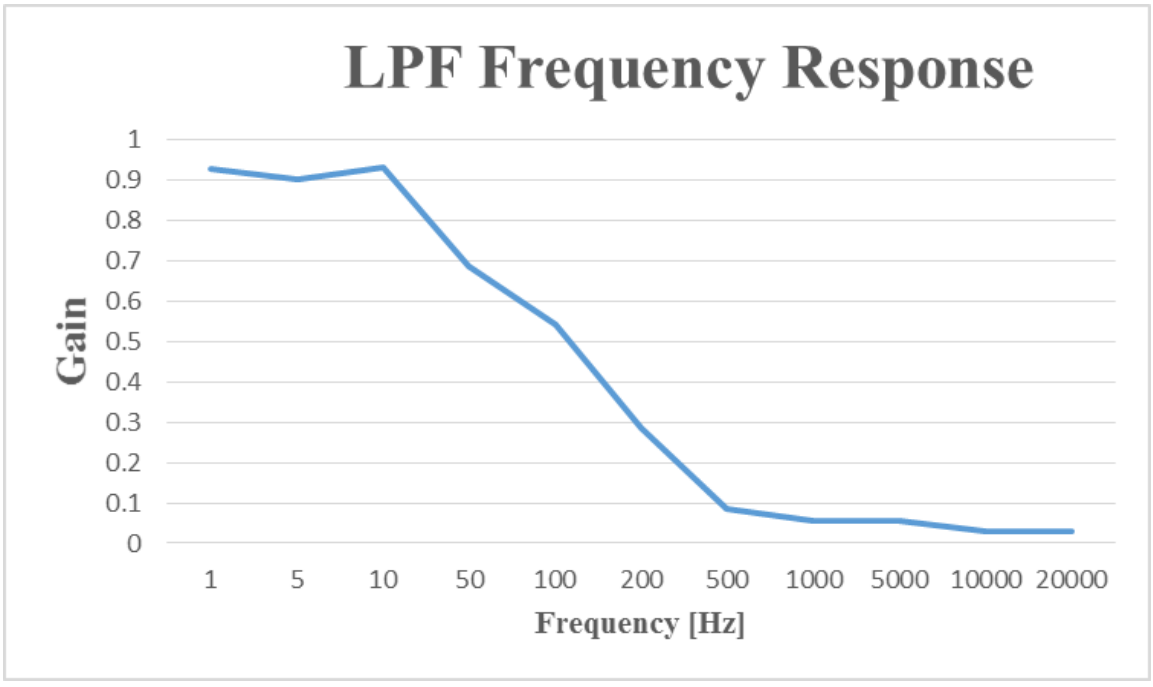

Figure 4.6: Plot of the LPF frequency response.

4.4: Accuracy:

Create a table showing the true temperature (x_{ti} as determined by the Fluke meter), and measured temperature (xmi using your device). Calculate average accuracy

True Temperature [C]: 35 ºC **Samples: 5**

Accuracy [C]: 0.096 ºC

Trial	Measured Temperature [C]
0	34.86
1	34.88
2	34.89
3	34.93
4	34.96

Figure 4.6: Accuracy Measurements. Poll the thermistor 5 times.

4.5: Reproducibility:

Record 10 independent temperature measurements. Calculate the standard deviation of these data and report S (estimation of σ) as reproducibility.

True Temperature [C]: 24.5 ºC **Samples: 10**

Reproducibility [C]: 0.043 ºC

Trial	Measured Temperature [C]
0	24.67
1	24.54
2	24.59
3	24.66
4	24.64
5	24.68
6	24.66
7	24.67
8	24.67
9	24.61

Figure 4.7: Reproducibility Measurements. Poll the thermistor 10 times.

5.0: Analysis and Discussion:

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

The Nyquist theorem generally states that you can only accurately represent signals that are bandlimited by ½ the original sampling frequency, f_s. Signals above the ½ f_s band limit are aliased, and they corrupt the signal that is measured by the sampling system.

Since Temperature only varies at approximately 10Hz, we need to sample above 20Hz to accurately represent the signal. Also, we need to use a low pass filter to eliminate the higher frequency signals that will alias and add noise to our temperature signal.

2) Explain the difference between resolution and accuracy.

Resolution is the smallest possible difference measurable by our data sampling system. Accuracy is the degree to which a measured actual value conforms to the expected theoretical value. A small resolution does not necessarily correspond to a high accuracy level. Accuracy is a sampling statistic while resolution is an inherent parameter to the design. The system can be accurate but not precise, precise but not accurate, neither, or both.

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

Precision can be thought of as the number of different representable outputs. Let p be the precision and dy be the resolution. Then, let y be the range of the system and r be the reproducibility. The distance of approximately 3 standard deviations covers the range, so:...

$$Y = p * dy = 3 * R$$

4) What is the purpose of the LPF?

The low pass filter serves to filter out signals with higher frequencies than our sampling frequency. This is to prevent higher frequency noise from aliasing our temperature signal input.

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

When we observe the R vs. T curve within a small temperature range (dT), the curve is approximately linear over this range. Thus, we can approximate the exponential function relatively well with a linear approximation if we restrict our system to operate in this range.

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 chose to use a timer-triggered ADC conversion process over the software-triggered alternative. Using the timer-triggered ADC conversion allows us to eliminate the possibility of sampling jitter. Thus, our samples are taken exactly $1/f_s$ intervals apart. Software triggered sampling often leads to a higher sampling jitter. However, software triggering would have been a simpler method to implement and would not have required us to understand the concept of output compare or fifo queues.