Stephen Hall and Razik Ahmed
Lab 07 TA: Harshad Desai

A) Objectives

Study ADC conversions, the Nyquist Theorem, the Valvano Postulate, and develop a temperature measurement system using a thermistor.

B) Hardware Design

The circuit diagram of the thermistor and LCD interface is on page 3.

C) Software Design (a hardcopy software printout is due at the time of demonstration)

1) Calibration data (procedure 5 and the **calib.h** file) – We did not do this.
2) Low level ADC interface (**ADC.c** and **ADC.h** files) – On later pages of this report.
3) Main program used to measure temperature – On later pages of this report.

D) Measurement Data

1) Sketch three waveforms (procedure 1)

The waveforms will be after the code pages.

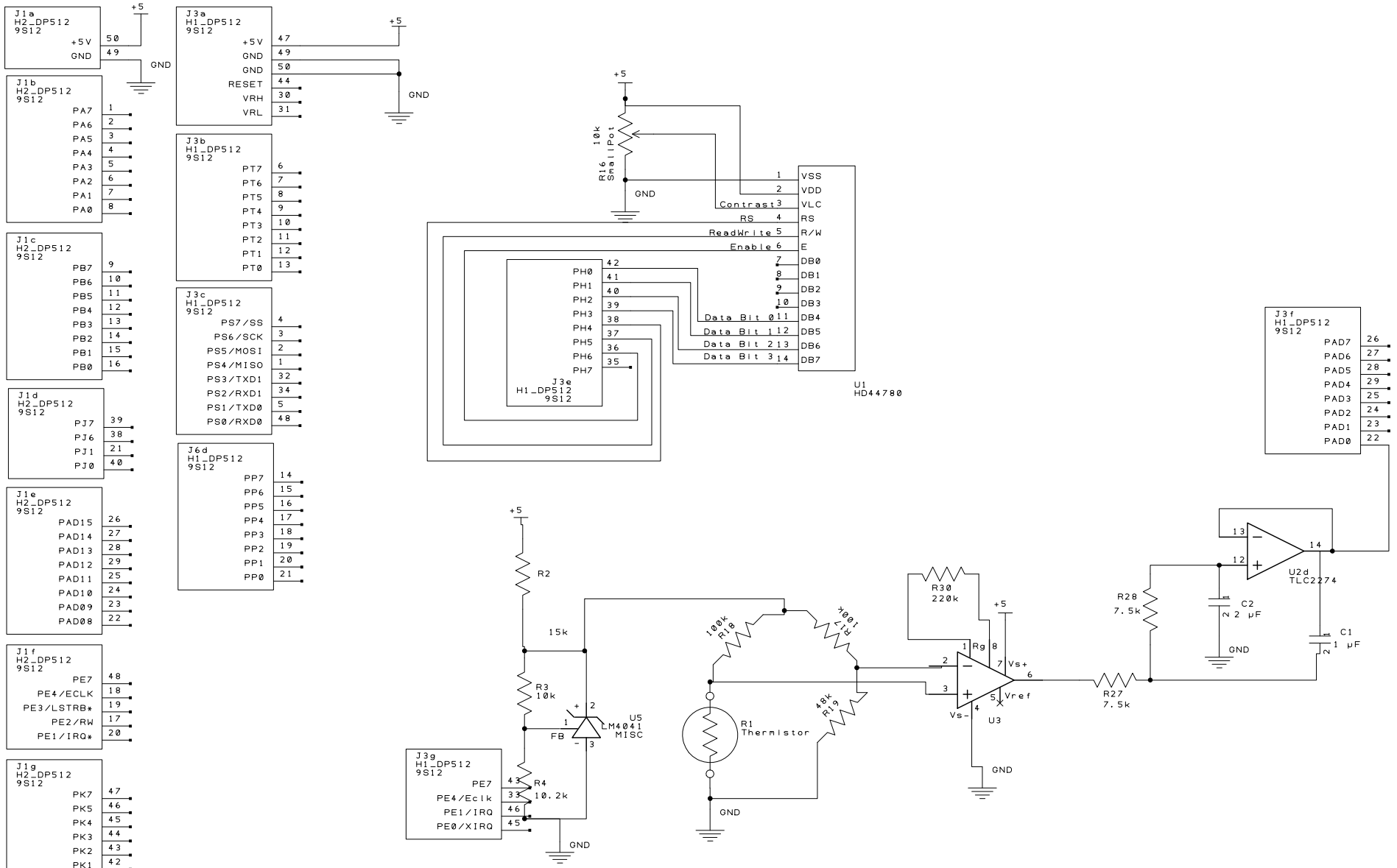
2) Static circuit performance (procedure 2,4)

The voltage at the top of the bridge is 2.49 V. The voltage across the 48k resistor is 0.807 V. The voltage out of the instrumentation amp is when the thermistor is shorted is 4.772. The gain of our instrumentation amp is 5.91. Using the LCD to display the voltage into the ADC we found that the voltage out of the amp at room temperature is around 2.61 V.

3) Dynamic circuit performance (procedure 3) – We did not do this.
4) Accuracy (procedure 6) - We did not do this.
5) Reproducibility (procedure 7) - We did not do this.

E) Analysis and Discussion (1 page maximum)

Hey Harshad, so there are no questions for this section in the lab manual. So we figured that we just could discuss this lab however we wanted. This lab was not fun. We did not like it. At all. The other labs were fun, unlike this one. We hope it dies. So we're probably missing a lot of stuff on this lab report. We know you told us when we were missing something in Lab 5, you probably don't need to let us know for this one. Just take off the points. Hopefully you don't think we're bad people, we just really didn't like this lab. We worked hard on all the other labs and will definitely work hard for the next 3. Using thermistors to measure temperature is a horrible idea. We can take the hit for this lab write-up as long as it doesn't affect any of our other grades. We decided to still work on Procedure 1 because it seemed more useful than the other parts of this lab because Valvano's Postulate intrigues us. We noticed that sampling 10 times faster gives a nice representation. Sampling at twice the frequency (500 Hz) gave two sine waves. If we took more samples it would be more apparent. Undersampling, we got a sine wave, but the frequency is not the correct frequency of the original wave. Like we said, intriguing.



All resistors are 1/4 watt 5% carbon composition
C1 is ceramic Z5U
Switches are B3F-1052 Digikey SW405-ND
Red LEDs, T1 3/4, 20mA Digikey 160-1087-ND
Yellow LEDs, T1 3/4, 20mA Digikey 160-1088-ND
Green LEDs, T1 3/4, 20mA Digikey 160-1089-ND
Slide pot, Alpha RA300BF-10-20D1-B54 Mouser# 312-9100F-50K

University Of Texas At Austin	
Schematic Name: Lab7 Thermistor	
Name(s): Stephen Hall and Razik Ahmed	
Date: Oct. 13, 2010	**Semester:** Fall 2010

A1

```
/*
  Initiating ADC Conversion:
    1. Writing to ATD0CTL5
    2. Edge on external trigger
    3. Level on external trigger

  Completed Conversion
    1. Reading ATD0STAT1
    2. Interrupt when complete
*/

#include <hidef.h>      /* common defines and macros */
#include <mc9s12dp512.h>     /* derivative information */
#pragma LINK_INFO DERIVATIVE "mc9s12dp512"

#define PROCEDURE 2
#include "PLL.h"
#include "ADC.h"
#include "OC.h"
#include "lcd.h"
#include "temperature.h"
#include <stdio.h>

#if PROCEDURE == 1
#include "SCI0.h"

unsigned short DataBuffer[100];
unsigned short Count=0;
void back(void) {
  unsigned short data;
  if(Count<100) {
    data = ADC0_In(0x80); // your program that samples channel 5
    DataBuffer[Count++] = data;
  }
}
void main(void){unsigned short i;
  PLL_Init(); // 24 MHz
  ADC0_Init(); // your module
  SCI0_Init(115200); // SCI output to PC
  OC0_Init(1000,&back); // your module sampling at 1000 Hz

  asm cli;
  while(Count<100) {}; // copy ADC to buffer in background
  for(i=0; i<100; i++) {
    SCI0_OutUDec(DataBuffer[i]); SCI0_OutChar(10);SCI0_OutChar(13);
  }
  for(;;){};
}

#else
#include "FIFO.h"

void getData(void) {
  unsigned short data;
  data = ADC0_In(0x82);
  while(!Fifo_Put(data)) {}
}

void main(void) {
  char buffer[10] = "";
  // Initialize needed modules
  DDRP |= 0x80;
  PLL_Init();
  Fifo_Init();
  ADC0_Init();
  OC0_Init(100, &getData);
  LCD_Open();

  LCD_Clear();
  sprintf(buffer, "     %cC", 223);
  LCD_OutString(buffer);

  asm cli

  for(;;) {
    unsigned short data;
    unsigned short temperature;
    while(!Fifo_Get(&data)) {}
```

```
 79:
 80:     temperature = Temp_Data(data);
 81:     sprintf(buffer, "%2d.%02d", temperature/100, temperature%100);
 82:     //sprintf(buffer, "%4d", data);
 83:     LCD_GoTo(0,0);
 84:     LCD_OutString(buffer);
 85:
 86:   }
 87: }
 88: #endif
```

```
1: #include <hidef.h>      /* common defines and macros */
2: #include <mc9s12dp512.h>     /* derivative information */
3: #pragma LINK_INFO DERIVATIVE "mc9s12dp512"
4:
5: unsigned short TempTable[];
6:
7: void ADC0_Init(void);
8: unsigned short ADC0_In(unsigned char channel);
```

```
 1: #include "ADC.h"
 2:
 3: void ADC0_Init(void) {
 4:   ATD0CTL2 = 0x80;
 5:   ATD0CTL3 = 0x08;
 6:   ATD0CTL4 = 0x05;
 7: }
 8:
 9: unsigned short ADC0_In(unsigned char channel) {
10:   unsigned short data;
11:   ATD0CTL5 = channel;
12:   while(!(ATD0STAT1&0x01)) {}
13:   data = ATD0DR0;
14:   return data;
15: }
```

```
1: // filename ************ Fifo.h ********
2: // Header file for the receive FIFO (two versions)
3: // Jonathan W. Valvano 10/1/07
4:
5: //  This example accompanies the books
6: //   "Embedded Microcomputer Systems: Real Time Interfacing",
7: //        Thompson, copyright (c) 2006,
8: //   "Introduction to Embedded Microcomputer Systems:
9: //    Motorola 6811 and 6812 Simulation", Brooks-Cole, copyright (c) 2002
10:
11: // Copyright 2006 by Jonathan W. Valvano, valvano@mail.utexas.edu
12: //    You may use, edit, run or distribute this file
13: //    as long as the above copyright notice remains
14:
15:
16:
17: #include <hidef.h>      /* common defines and macros */
18: #include <mc9s12dp512.h>     /* derivative information */
19: #pragma LINK_INFO DERIVATIVE "mc9s12dp512"
20:
21: #define FIFOSIZE 24
22: /* Number of characters in the Fifo
23:     the FIFO is full when it has FifoSize-1 characters */
24: void Fifo_Init(void);
25:
26: /*-----------------------Fifo_Put---------------------------
27:   Enter one character into the fifo
28:   Inputs: 8-bit data
29:   Outputs: true if data is properly saved,
30:            false if data not saved because it was previously full*/
31: int Fifo_Put(unsigned short data);
32:
33: /*-----------------------Fifo_Get---------------------------
34:   Remove one character from the fifo
35:   Inputs: pointer to place to return 8-bit data
36:   Outputs: true if data is valid,
37:            false if fifo was empty at the time of the call*/
38: int Fifo_Get(unsigned short *datapt);
39:
40: /*-----------------------Fifo_Status---------------------------
41:   Check the status of the fifo
42:   Inputs: none
43:   Outputs: true if there is any data in the fifo */
44: int Fifo_Status(void);
45:
```

```
 1: // filename ************Fifo.c********
 2: // Two implementations of a FIFO
 3: // Jonathan W. Valvano 10/1/07
 4:
 5: //  This example accompanies the books
 6: //   "Embedded Microcomputer Systems: Real Time Interfacing",
 7: //        Thompson, copyright (c) 2006,
 8: //   "Introduction to Embedded Microcomputer Systems:
 9: //    Motorola 6811 and 6812 Simulation", Brooks-Cole, copyright (c) 2002
10:
11: // Copyright 2007 by Jonathan W. Valvano, valvano@mail.utexas.edu
12: //    You may use, edit, run or distribute this file
13: //    as long as the above copyright notice remains
14:
15: #include "Fifo.h"
16:
17: /* Number of characters in the Fifo
18:      the FIFO is full when it has FifoSize-1 characters */
19: unsigned short static volatile *PutPt;    /* Pointer of where to put next */
20: unsigned short static volatile *GetPt;    /* Pointer of where to get next */
21:                  /* FIFO is empty if PutPt=GetPt */
22:                  /* FIFO is full  if PutPt+1=GetPt */
23: unsigned short static Fifo[FIFOSIZE];     /* The statically allocated fifo data */
24:
25: /*----------------------Fifo_Init---------------------------
26:   Initialize fifo to be empty
27:   Inputs: none
28:   Outputs: none */
29: void Fifo_Init(void){
30:   unsigned char SaveCCR;
31:   asm  tpa
32:   asm  staa SaveCCR
33:   asm  sei          // make atomic
34:   PutPt=GetPt=&Fifo[0];   // Empty when PutPt=GetPt
35:   asm  ldaa SaveCCR
36:   asm  tap          // end critical section
37: }
38:
39: /*----------------------Fifo_Put---------------------------
40:   Enter one character into the fifo
41:   Inputs: 8-bit data
42:   Outputs: true if data is properly saved,
43:            false if data not saved because it was previously full*/
44: int Fifo_Put(unsigned short data){
45: unsigned short volatile *tempPt;
46:   tempPt = PutPt;
47:   *(tempPt) = data;      // try to Put data into fifo
48:   tempPt++;
49:   if(tempPt == &Fifo[FIFOSIZE]){ // need to wrap?
50:     tempPt = &Fifo[0];
51:   }
52:   if(tempPt == GetPt){
53:     return(0);        // Failed, fifo was previously full
54:   }
55:   else{
56:     PutPt = tempPt;  // Success, so update pointer
57:     return(1);
58:   }
59: }
60:
61: /*----------------------Fifo_Get---------------------------
62:   Remove one character from the fifo
63:   Inputs: pointer to place to return 8-bit data
64:   Outputs: true if data is valid,
65:            false if fifo was empty at the time of the call*/
66: int Fifo_Get(unsigned short *datapt){
67:   if(PutPt == GetPt){
68:     return(0);     // Empty if PutPt=GetPt
69:   }
70:   else{
71:     *datapt = *(GetPt);  // return by reference
72:     GetPt++;             // removes data from fifo
73:     if(GetPt == &Fifo[FIFOSIZE]){
74:       GetPt = &Fifo[0];  // wrap
75:     }
76:     return(1);
77:   }
78: }
```

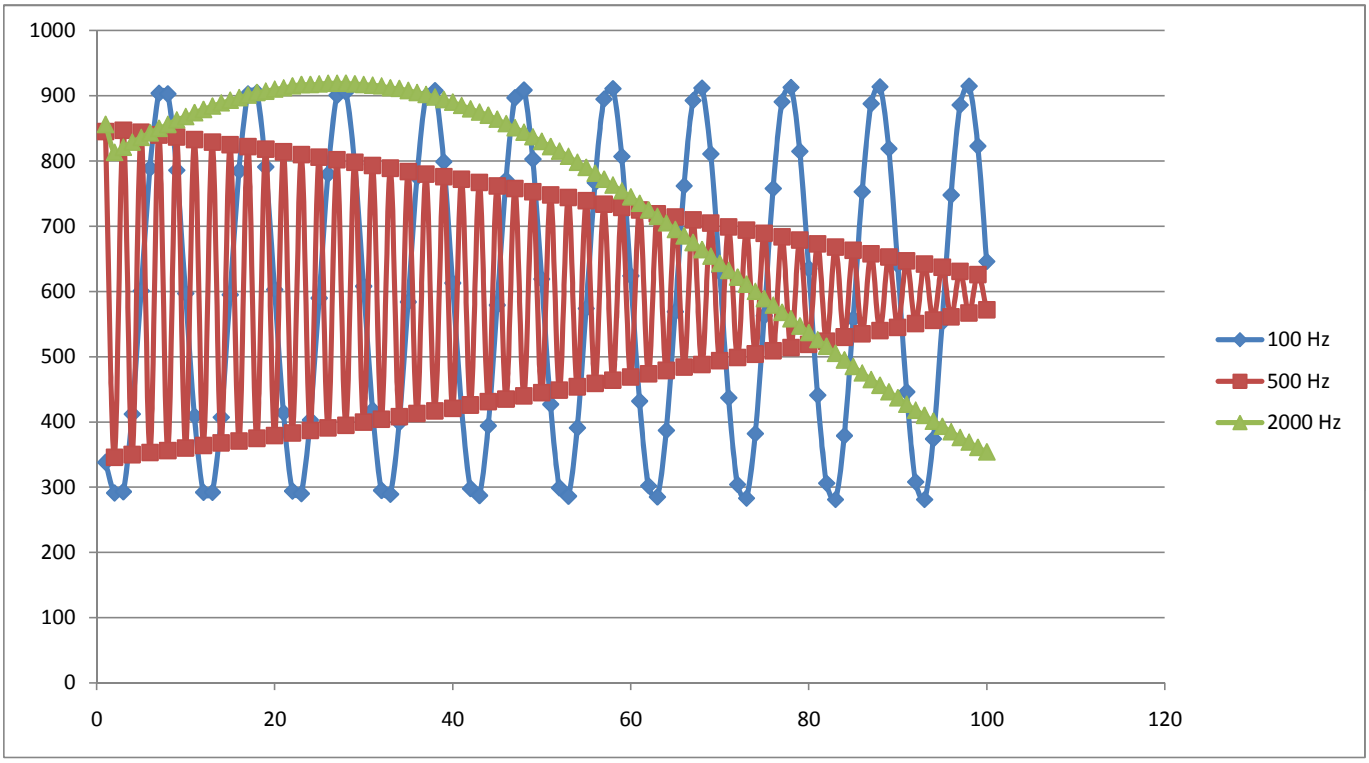
```
79:
80: /*-----------------------Fifo_Status----------------------------
81:   Check the status of the fifo
82:   Inputs: none
83:   Outputs: true if there is any data in the fifo */
84: int Fifo_Status(void){
85:   return (PutPt != GetPt);
86: }
```

```
1: #include <hidef.h>      /* common defines and macros */
2: #include <mc9s12dp512.h>     /* derivative information */
3: #pragma LINK_INFO DERIVATIVE "mc9s12dp512"
4:
5: void OC0_Init(unsigned short freq, void (*func)(void));
```

```
 1: #include "OC.h"
 2:
 3: unsigned short frequency;
 4: void (*function)(void);
 5:
 6: interrupt 8 void TOC0handler(void){ // executes at 100 Hz
 7:   TFLG1 = 0x01;         // acknowledge OC0
 8:   PTP |= 0x80;
 9:   (*function)();
10:   TC0 += 187500/frequency;
11: }
12:
13: //---------------------OC_Init0---------------------
14: // arm output compare 0 for 100Hz periodic interrupt
15: // Input: none
16: // Output: none
17: void OC0_Init(unsigned short freq, void (*func)(void)) {
18:   frequency = freq;
19:   function = func;
20:   TIOS |= 0x01;   // activate TC0 as output compare
21:   TIE  |= 0x01;   // arm OC0
22:   TSCR1 = 0x80;   // Enable TCNT, 24MHz boot mode, 8MHz in run mode
23:   TSCR2 = 0x07;   // divide by 64 TCNT prescale, TOI disarm
24:   PACTL = 0;      // timer prescale used for TCNT
25:   TC0   = TCNT + 50; // first interrupt right away
26: }
```

```
1: unsigned short Temp_Data(unsigned short adc);
```

```
 1: unsigned short const ADCdata[53]={0,60,72,85,97,110,122,135,148,161,174,
 2:      187,201,214,227,241,255,268,282,296,310,
 3:      324,338,352,366,381,395,409,424,438,453,
 4:      468,482,497,512,527,541,556,571,586,601,
 5:      616,631,646,662,677,692,707,722,737,752,767,1024};
 6:
 7:
 8: unsigned short const Tdata[53]={4000,4000,3960,3920,3880,3840,3800,3760,3720,3680,3640,
 9:      3600,3560,3520,3480,3440,3400,3360,3320,3280,3240,
10:      3200,3160,3120,3080,3040,3000,2960,2920,2880,2840,
11:      2800,2760,2720,2680,2640,2600,2560,2520,2480,2440,
12:      2400,2360,2320,2280,2240,2200,2160,2120,2080,2040,2000,2000};
13:
14:
15: unsigned short Temp_Data(unsigned short adc) {
16:   unsigned short temp;
17:
18:   asm ldd adc
19:   asm Lookup: ldx  #ADCdata  // first find x1<=xL<x2
20:   asm ldy  #Tdata
21:   asm lookx1: cpd  2,x      // check xL<x2
22:   asm blo  found    // stops when X points to x1
23:   asm leax 2,x
24:   asm leay 2,y
25:   asm bra  lookx1
26:   asm found:   subd 0,x      // xL-x1
27:   asm pshd
28:   asm ldd  2,x      // x2
29:   asm subd 0,x      // D=x2-x1
30:   asm tfr  D,X      // X=x2-x1
31:   asm puld          // D=(xL-x1)
32:   asm fdiv          // X=(65536*(xL-x1))/(x2-x1)
33:   asm tfr  X,D
34:   asm tfr  A,B
35:   // B=(256*(xL-x1))/(x2-x1)
36:   // Y=>y1,y2
37:   asm etbl 0,y
38:   asm std temp
39:
40:    return temp;
41: }
```


1000
900
800
700
600
500
400
300
200
100
0
0
20
40
60
80
100
120
100 Hz
500 Hz
2000 Hz