AN1315
Rev 2, 05/2005

An Evaluation System Interfacing the MPX2000 Series Pressure Sensors to a Microprocessor

by: **Bill Lucas**
Discrete Applications Engineering

INTRODUCTION

Outputs from compensated and calibrated semiconductor pressure sensors such as the MPX2000 series devices are easily amplified and interfaced to a microprocessor. Design considerations and the description of an evaluation board using a simple analog interface connected to a microprocessor is presented here.

PURPOSE

The evaluation system shown in Figure 1 shows the ease of operating and interfacing the Freescale Semiconductor, Inc. MPX2000 series pressure sensors to a quad operational amplifier, which amplifies the sensor's output to an acceptable level for an analog-to-digital converter. The output of the op amp is connected to the A/D converter of the microprocessor and that analog value is then converted to engineering units and displayed on a liquid crystal display (LCD). This system may be used to evaluate any of the MPX2000 series pressure sensors for your specific application.

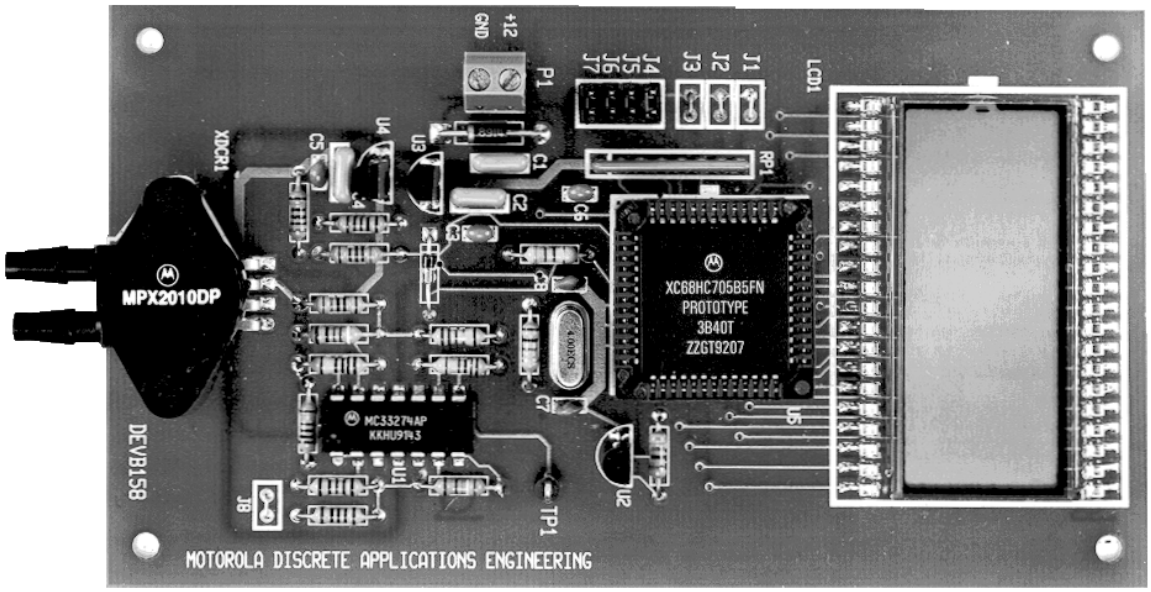

Figure 1. DEVB158 2000 Series LCD Pressure Gauge EVB (Board No Longer Available)

DESCRIPTION

The DEVB158 evaluation system is constructed on a small printed circuit board. Designed to be powered from a 12 Vdc power supply, the system will display the pressure applied to the MPX2000 series sensor in pounds per square inch (PSI) on the liquid crystal display. Table 1 shows the pressure sensors that may be used with the system and the pressure range associated with that particular sensor as well as the jumper configuration required to support that sensor. These jumpers are installed at assembly time to correspond with the supplied sensor. Should the user chose to evaluate a different sensor other than that supplied with the board, the jumpers must be changed to correspond to Table 1 for the new sensor. The displayed pressure is scaled to the full scale (PSI) range of the installed pressure sensor. No potentiometers are used in the system to adjust its span and offset. This function is performed by software.

Table 1. Missing Table Head

Sensor Type	Input Pressure PSI	Jumpers			
		J8	J3	J2	J1
MPX2010	0-1.5	IN	IN	IN	IN
MPX2050	0-7.5	OUT	IN	IN	OUT
MPX2100	0-15.0	OUT	IN	OUT	IN
MPX2200	0-30	OUT	IN	OUT	OUT

The signal conditioned sensor's zero pressure offset voltage with no pressure applied to the sensor is empirically computed each time power is applied to the system and stored in RAM. The sensitivity of the MPX2000 series pressure sensors is quite repeatable from unit to unit. There is a facility for a small adjustment of the slope constant built into the program. It is accomplished via jumpers J4 through J7, and will be explained in the OPERATION section.

Figure 2 shows the printed circuit silkscreen and Figure 3 and Figure 4show the schematic for the system.

The analog section of the system can be broken down into two subsections. These sections are the power supply and the amplification section. The power supply section consists of a diode, used to protect the system from input voltage reversal, and two fixed voltage regulators. The 5 volt regulator (U3) is used to power the microprocessor and display. The 8 volt regulator (U4) is used to power the pressure sensor, voltage references and a voltage offset source.

The microprocessor section (U5) requires minimal support hardware to function. The MC34064P-5 (U2) provides an under voltage sense function and is used to reset the microprocessor at system power-up. The 4.0 MHz crystal (Y1) provides the external portion of the oscillator function for clocking the microprocessor and providing a stable base for timing functions.

The analog section of the system can be broken down into two subsections. These sections are the power supply and the amplification section. The power supply section consists of a diode, used to protect the system from input voltage reversal, and two fixed voltage regulators. The 5 volt regulator (U3) is used to power the microprocessor and display. The 8 volt regulator (U4) is used to power the pressure sensor, voltage references and a voltage offset source.

The microprocessor section (U5) requires minimal support hardware to function. The MC34064P-5 (U2) provides an under voltage sense function and is used to reset the microprocessor at system power-up. The 4.0 MHz crystal (Y1) provides the external portion of the oscillator function for clocking the microprocessor and providing a stable base for timing functions.

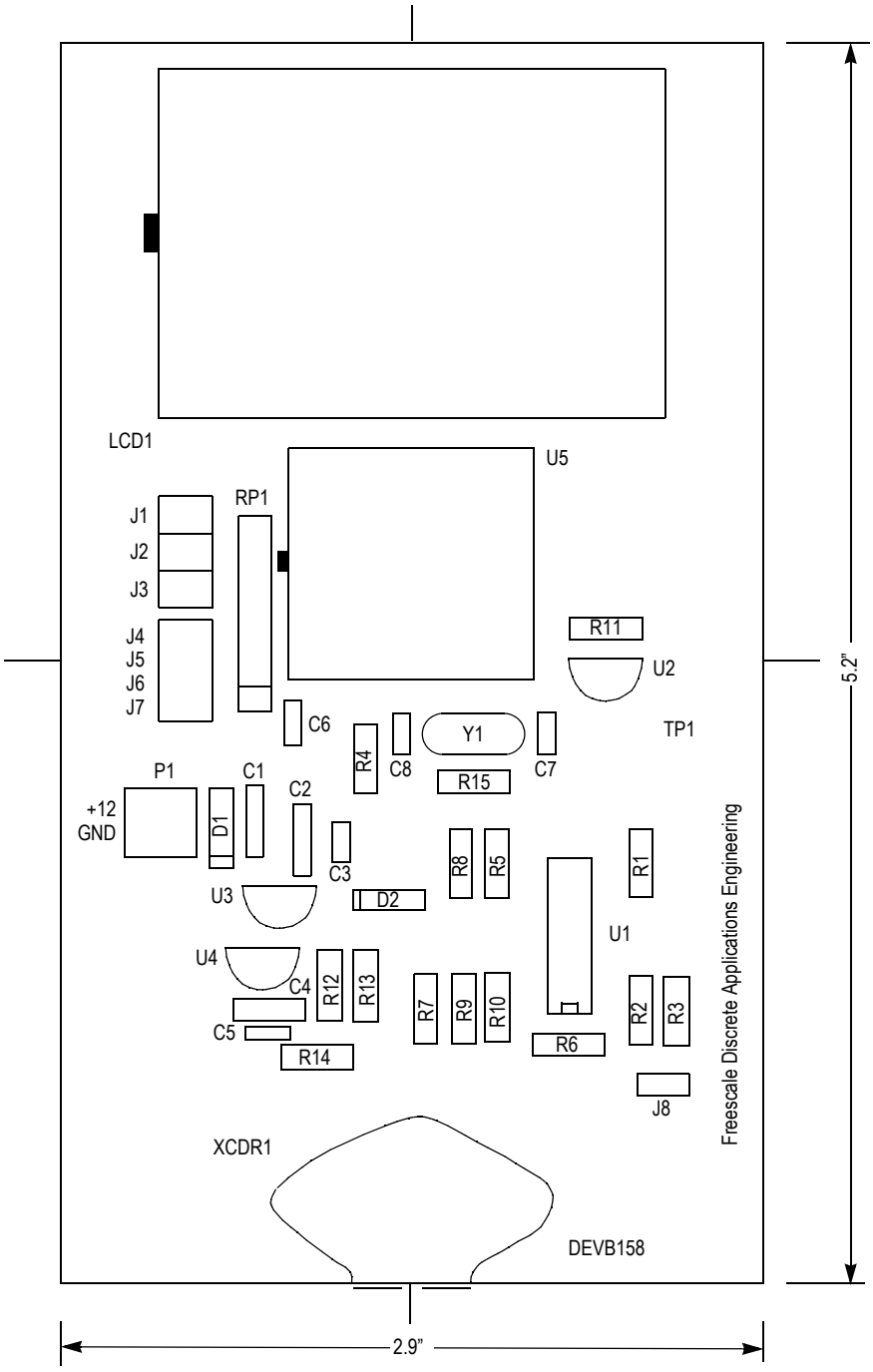

Figure 2. Printed Circuit Silkscreen

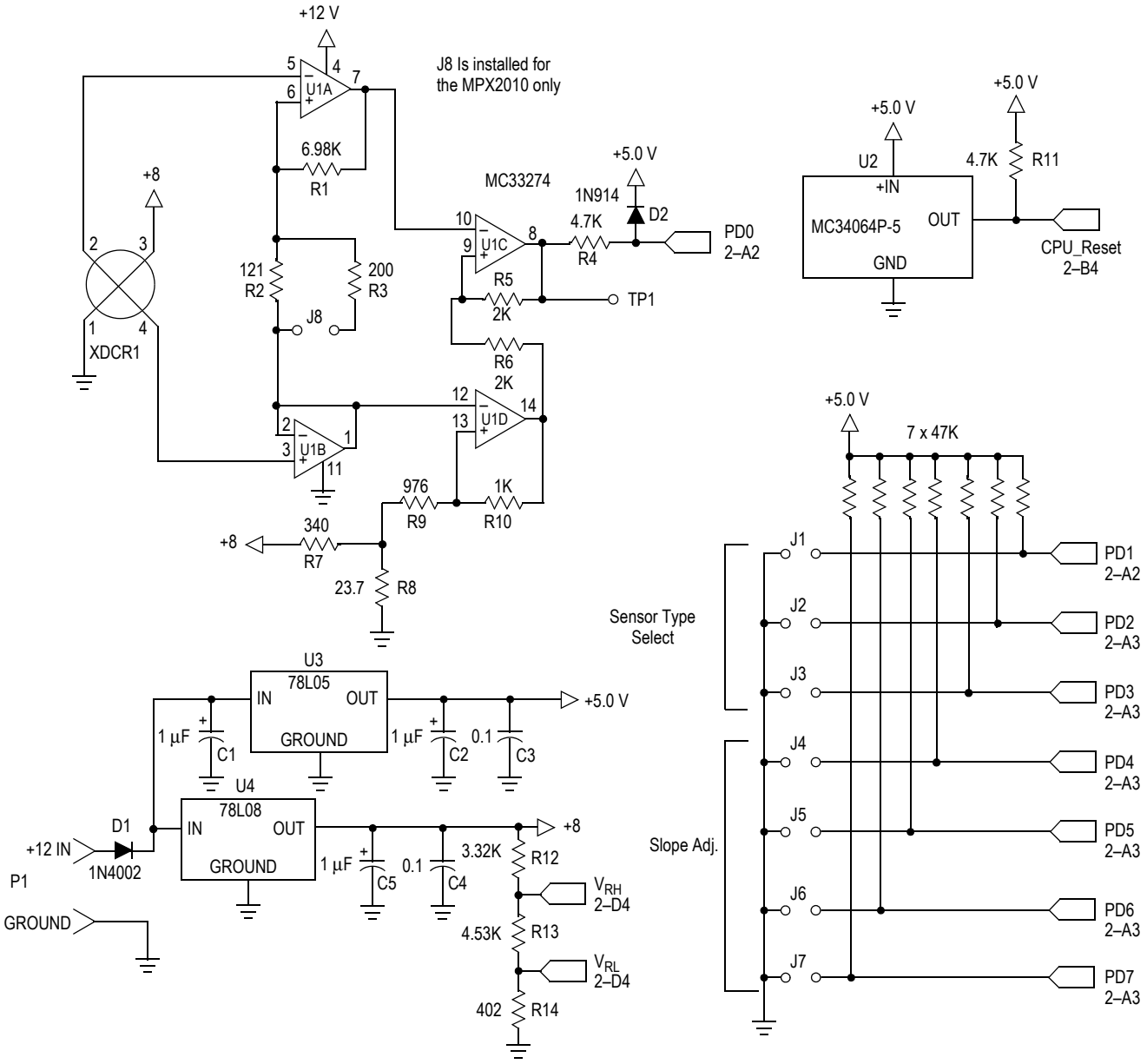

Figure 3. Schematic A

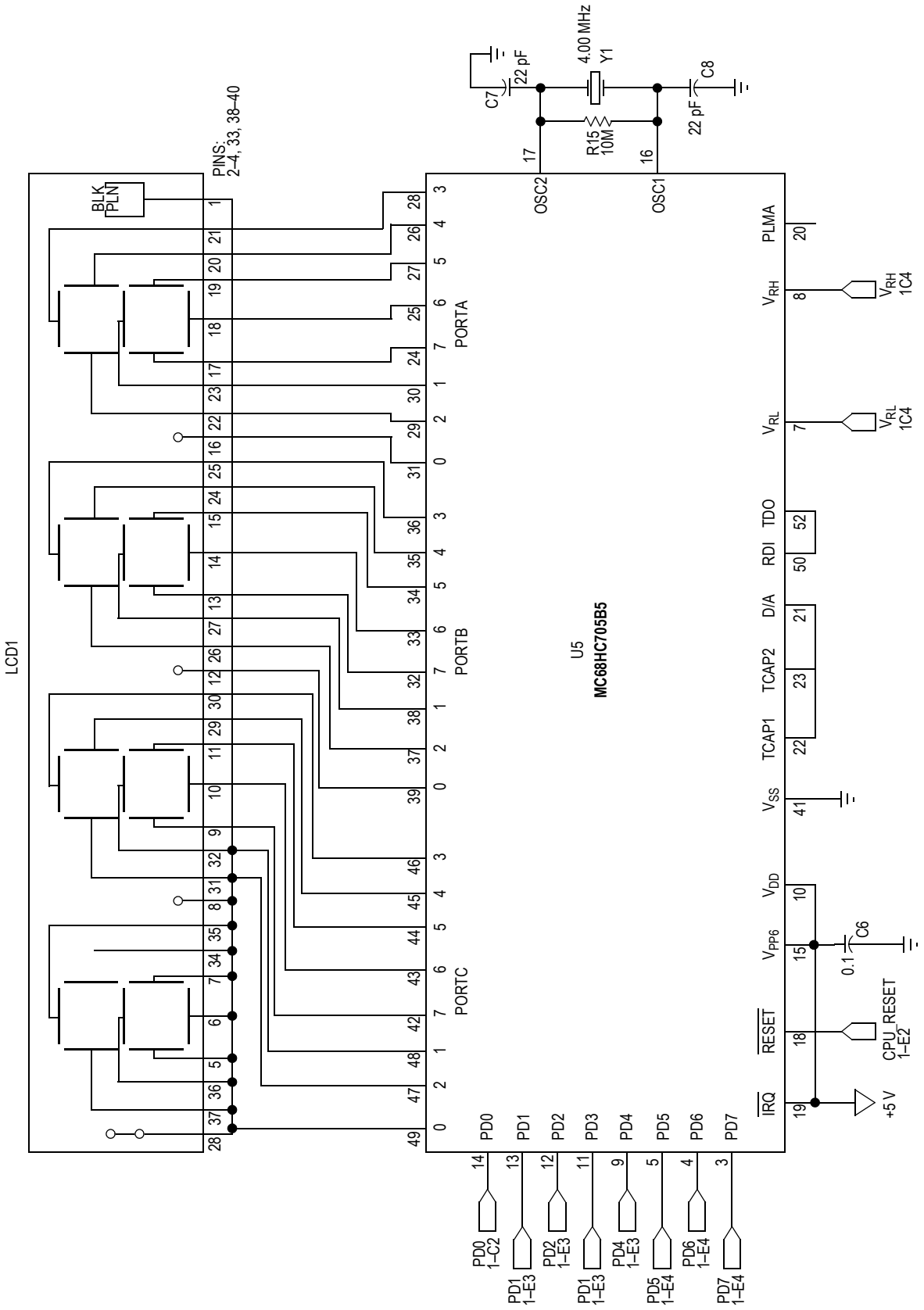

Figure 4. Schematic B

Table 2. Parts List

Designators	Quant.	Description	Rating	Manufacturer	Part Number
C3, C4, C6	3	0.1 µF Ceramic Cap.	50 Vdc	Sprague	1C105Z5U104M050B
C1, C2, C5	3	1 µF Ceramic Cap.	50 Vdc	muRATA ERIE	RPE123Z5U105M050V
C7, C8	2	22 pF Ceramic Cap.	100 Vdc	Mepco/Centralab	CN15A220K
J1-J3, J8	3 OR 4	#22 or #24 AWG Tined Copper		As Required	
J4-J7	1	Dual Row Straight 4 Pos. Arranged On 0.1″ Grid		AMP	87227-2
LCD1	1	Liquid Crystal Display		IEE	LCD5657
P1	1	Power Connector		Phoenix Contact	MKDS 1/2-3.81
R1	1	6.98K Ohm resistor 1%			
R2	1	121 Ohm Resistor 1%			
R3	1	200 Ohm Resistor 1%			
R4, R11	2	4.7K Ohm Resistor			
R7	1	340 Ohm Resistor 1%			
R5, R6	2	2.0K Ohm Resistor 1%			
R8	1	23.7 Ohm Resistor 1%			
R9	1	976 Ohm Resistor 1%			
R10	1	1K Ohm Resistor 1%			
R12	1	3.32K Ohm Resistor 1%			
R13	1	4.53K Ohm Resistor 1%			
R14	1	402 Ohm Resistor 1%			
R15	1	10 Meg Ohm Resistor			
RP1	1	47K Ohm x 7 SIP Resistor 2%		CTS	770 Series
TP1	1	Test Point	Red	Components Corp.	TP-104-01-02
U1	1	Quad Operational Amplifier		Freescale	MC33274P
U2	1	Under Voltage Detector		Freescale	MC34064P-5
U3	1	5 Volt Fixed Voltage Regulator		Freescale	MC78L05ACP
U4	1	8 Volt Fixed Voltage Regulator		Freescale	MC78L08ACP
U5	1	Microprocessor		Freescale Freescale	MC68HC705B5FN or XC68HC705B5FN
XDCR	1	Pressure Sensor		Freescale	MPX2xxxDP
Y1	1	Crystal (Low Profile)	4.0 MHz	CTS	ATS040SLV
No Designator	1	52 Pin PLCC Socket for U5		AMP	821-575-1
No Designator	4	Jumpers For J4 thru J7		Molex	15-29-1025
No Designator	1	Bare Printed Circuit Board			
No Designator	4	Self Sticking Feet		Fastex	5033-01-00-5001

Notes: All resistors are 1/4 W resistors with a tolerance of 5% unless otherwise noted.
All capacitors are 100 volt, ceramic capacitors with a tolerance of 10% unless otherwise noted.

OPERATIONAL CHARACTERISTICS

The following operational characteristics are included as a guide to operation.

Characteristic	Symbol	Min	Max	Unit
Power Supply Voltage	+12	10.75	16	Volts
Operating Current	I_{CC}		75	mA
Full Scale Pressure	P_{fs}			
MPX2010			1.5	PSI
MPX2050			7.5	PSI
MPX2100			15	PSI
MPX2200			30	PSI

Pin-by-Pin Description

+12

Input power is supplied at the +12 terminal. The minimum operating voltage is 10.75 Vdc and the maximum operating voltage is 16 Vdc.

GND

The ground terminal is the power supply return for the system.

TP1

Test point 1 is connected to the final op amp stage. It is the voltage that is applied to the microprocessor's A/D converter.

There are two ports on the pressure sensor located at the bottom center of the printed circuit board. The pressure port is on the top left and the vacuum port is on the bottom right of the sensor.

OPERATION

Connect the system to a 12 Vdc regulated power supply. (Note the polarity marked on the power terminal P1.) Depending on the particular pressure sensor being used with the system, wire jumpers J1 through J3 and J8 must be installed at board assembly time. If at some later time it is desirable to change the type of sensor that is installed on the board, jumpers J1 through J3 and J8, must be reconfigured for the system to function properly (see Table 1). If an invalid J1 through J3 jumper combination (i.e., not listed in Table 1) is used the LCD will display “SE” to indicate that condition. These jumpers are read by the software and are used to determine which sensor is installed on the board. Wire jumper J8 is installed only when an MPX2010DP pressure sensor is used on the system. The purpose of wire jumper J8 will be explained later in the text. Jumpers J4 through J7 are read by the software to allow the user to adjust the slope constant used for the engineering units calculation (see Table 3). The pressure and vacuum ports on the sensor must be left open to atmosphere anytime the board is powered-up. This is because the zero pressure offset voltage is computed at power-up.

When you apply power to the system, the LCD will display CAL for approximately 5 seconds. After that time, pressure or vacuum may be applied to the sensor. The system will then start displaying the applied pressure in PSI.

Table 3. Slope Constants

J7	J6	J5	J4	Action
IN	IN	IN	IN	Normal Slope
IN	IN	IN	OUT	Decrease the Slope Approximately 7%
IN	IN	OUT	IN	Decrease the Slope Approximately 6%
IN	IN	OUT	OUT	Decrease the Slope Approximately 5%
IN	OUT	IN	IN	Decrease the Slope Approximately 4%
IN	OUT	IN	OUT	Decrease the Slope Approximately 3%
IN	OUT	OUT	IN	Decrease the Slope Approximately 2%
IN	OUT	OUT	OUT	Decrease the Slope Approximately 1%
OUT	IN	IN	IN	Increase the Slope Approximately 1%
OUT	IN	IN	OUT	Increase the Slope Approximately 2%
OUT	IN	OUT	IN	Increase the Slope Approximately 3%
OUT	IN	OUT	OUT	Increase the Slope Approximately 4%
OUT	OUT	IN	IN	Increase the Slope Approximately 5%
OUT	OUT	IN	OUT	Increase the Slope Approximately 6%
OUT	OUT	OUT	IN	Increase the Slope Approximately 7%
OUT	OUT	OUT	OUT	Normal Slope

To improve the accuracy of the system, you can change the constant used by the program that determines the span of the sensor and amplifier. You will need an accurate test gauge (using PSI as the reference) to measure the pressure applied to the sensor. Anytime after the display has completed the zero calculation, (after CAL is no longer displayed) apply the sensor's full scale pressure (see Table 1), to the sensor. Make sure that jumpers J4 through J7 are in the “normal” configuration (see Table 3). Referring to Table 3, you can better “calibrate” the system by changing the configuration of J4 through J7. To “calibrate” the system, compare the display reading against that of the test gauge (with J4 through J7 in the “normal slope” configuration). Change the configuration of J4 through J7 according to Table 3 to obtain the best results. The calibration jumpers may be changed while the system is powered up as they are read by the software before each display update.

DESIGN CONSIDERATIONS

To build a system that will show how to interface an MPX2000 series pressure sensor to a microprocessor, there are two main challenges. The first is to take a small differential signal produced by the sensor and produce a ground referenced signal of sufficient amplitude to drive a microprocessor's A/D input. The second challenge is to understand the microprocessor's operation and to write software that makes the system function.

From a hardware point of view, the microprocessor portion of the system is straight forward. The microprocessor needs power, a clock source (crystal Y1, two capacitors and a resistor), and a reset signal to make it function. As for the A/D converter, external references are required to make it function. In this case, the power source for the sensor is divided to produce the voltage references for the A/D converter. Accurate results will be achieved since the output from the sensor and the A/D references are ratiometric to its power supply voltage.

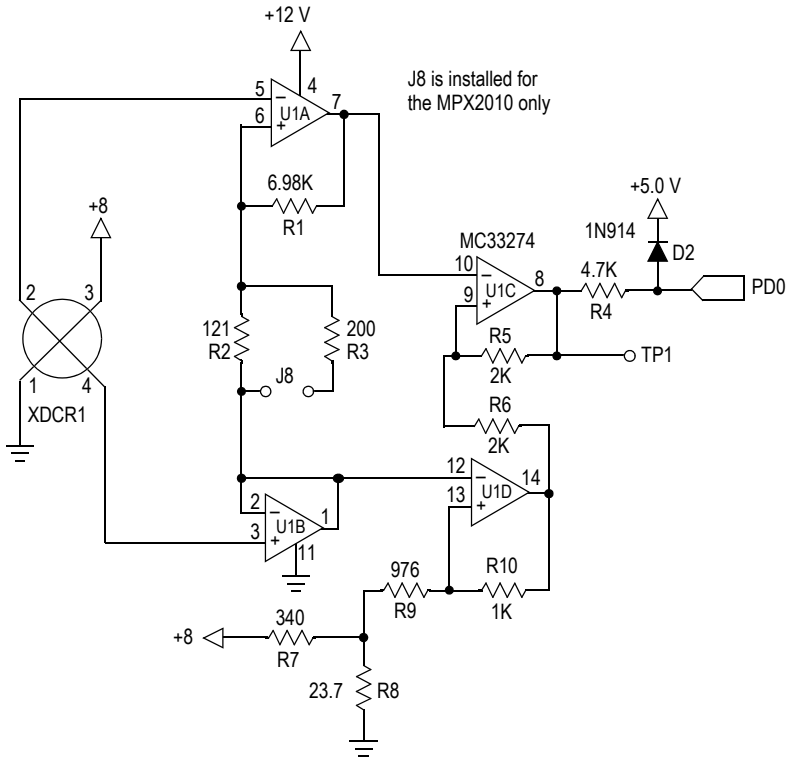

Figure 5. Analog Interface

The liquid crystal display is driven by Ports A, B and C of the microprocessor. There are enough I/O lines on these ports to provide drive for three full digits, the backplane and two decimal points. Software routines provide the AC waveform necessary to drive the display.

The analog portion of the system consists of the pressure sensor, a quad operational amplifier and the voltage references for the microprocessor's A/D converter and signal conditioning circuitry. Figure 5 shows an interface circuit that will provide a single ended signal with sufficient amplitude to drive the microprocessor's A/D input. It uses a quad operational amplifier and several resistors to amplify and level shift the sensor's output. It is necessary to level shift the output from the final amplifier into the A/D. Using single power supplied op amps, the V_{CE} saturation of the output from an op amp cannot be guaranteed to pull down to zero volts. The analog design shown here will provide a signal to the A/D converter with a span of approximately 4 volts when zero to full-scale pressure is applied to the sensor. The final amplifier's output is level shifted to approximately 0.7 volts. This will provide a signal that will swing between approximately 0.7 volts and 4.7 volts. The offset of 0.7 volts in this implementation does not have to be trimmed to an exact point. The software will sample the voltage applied to the A/D converter at initial power up time and call that value “zero”. The important thing to remember is that the span of the signal will be approximately 4 volts when zero to full scale pressure is applied to the sensor. The 4 volt swing in signal may vary slightly from sensor to sensor and can also vary due to resistor tolerances in the analog circuitry. Jumpers J4 through J7 may be placed in various configurations to compensate for these variations (see Table 3).

Referring to Figure 5, most of the amplification of the voltage from the pressure sensor is provided by U1A which is

configured as a differential amplifier. U1B serves as a unity gain buffer in order to keep any current that flows through R2 (and R3) from being fed back into the sensor's negative output. With zero pressure applied to the sensor, the differential voltage from pin 2 to pin 4 of the sensor is zero or very close to zero volts. The common mode, or the voltage measured between pins 2 or 4 to ground, is equal to approximately one half of the voltage applied to the sensor, or 4 volts. The zero pressure output voltage at pin 7 of U1A will then be 4 volts because pin 1 of U1B is also at 4 volts, creating a zero bias between pins 5 and 6 of U1A. The four volt zero pressure output will then be level shifted to the desired zero pressure offset voltage (approximately 0.7 volts) by U1C and U1D.

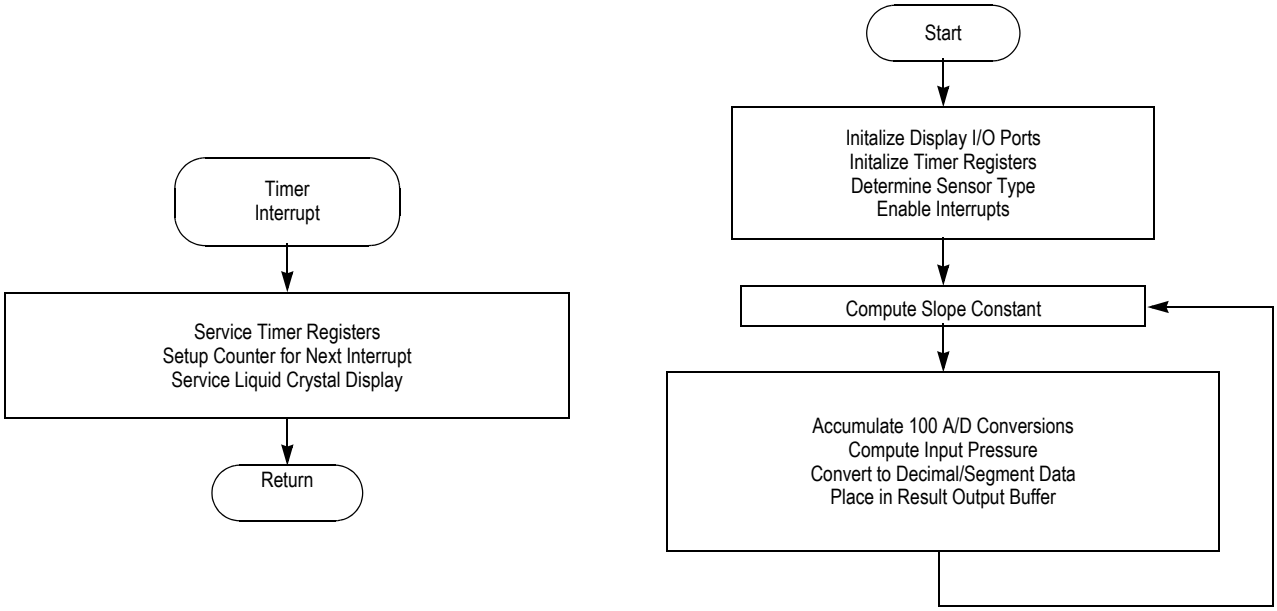

Figure 6. DEVB-158 Software Flowchart

To further explain the operation of the level shifting circuitry, refer again to Figure 5. Assuming zero pressure is applied to the sensor and the common mode voltage from the sensor is 4 volts, the voltage applied to pin 12 of U1D will be 4 volts, implying pin 13 will be at 4 volts. The gain of amplifier U1D will be (R10/(R8+R9)) +1 or a gain of 2. R7 will inject a V_{offset} (0.7 volts) into amplifier U1D, thus causing the output at U1D pin 14 to be 7.3 = (4 volts @ U1D pin 12 $\times$ 2) - 0.7 volts. The gain of U1C is also set at 2 ((R5/R6)+1). With 4 volts applied to pin 10 of U1C, its output at U1C pin 8 will be 0.7 = ((4 volts @ U1C pin 10 $\times$ 2) - 7.3 volts). For this scheme to work properly, amplifiers U1C and U1D must have a gain of 2 and the output of U1D must be shifted down by the V_{offset} provided by R7. In this system, the 0.7 volts V_{offset} was arbitrarily picked and could have been any voltage greater than the V_{sat} of the op amp being used. The system software will take in account any variations of V_{offset} as it assumes no pressure is applied to the sensor at system power up.

The gain of the analog circuit is approximately 117. With the values shown in Figure 5, the gain of 117 will provide a span of approximately 4 volts on U1C pin 8 when the pressure sensor and the 8 volt fixed voltage regulator are at their maximum output voltage tolerance. All of the sensors listed in Table 1 with the exception of the MPX2010DP output approximately 33 mV when full scale pressure is applied. When the MPX2010DP sensor is used, its full scale sensor differential output is approximately 20 mV. J8 must be installed to increase the gain of the analog circuit to still provide the 4 volts span out of U1C pin 8 with a 20 mV differential from the sensor.

Diode D2 is used to protect the microprocessor's A/D input if the output from U1C exceeds 5.6 volts. R4 is used to provide current limiting into D4 under failure or overvoltage conditions.

SOFTWARE

The source code, compiled listing, and S-record output for the software used in this system are available on the Freescale Freeware Bulletin Board Service in the MCU directory under the filename DEVB158.ARC. To access the bulletin board, you must have a telephone line, a 300, 1200 or 2400 baud modem and a personal computer. The modem must be compatible with the Bell 212A standard. Call (512) 891-3733 to access the Bulletin Board Service.

Figure 6 is a flowchart for the program that controls the system. The software for the system consists of a number of modules. Their functions provide the capability for system calibration as well as displaying the pressure input to the MPX2000 series pressure sensor.

The “C” compiler used in this project was provided by BYTE CRAFT LTD. (519) 888-6911. A compiler listing of the program is included at the end of this document. The following is a brief explanation of the routines:

delay() Used to provide a software loop delay.

read_a2d() Performs 100 reads on the A/D converter on multiplexer channel 0 and returns the accumulation.

fixcompare() Services the internal timer for 15 ms. timer compare interrupts.

TIMERCMP() Alternates the data and backplane inputs to the liquid crystal display.

initio() Sets up the microprocessor's I/O ports, timer and enables processor interrupts.

adzero() This routine is called at powerup time. It delays to let the power supply and the transducer stabilize. It then calls “read_atod()” and saves the returned value as the sensors output voltage with zero pressure applied.

cvt_bin_dec(unsigned long arg This routine converts the unsigned binary argument passed in “arg” to a five digit decimal number in an array called “digit.” It then uses the decimal results for each digit as an index into a table that converts the decimal number into a segment pattern for the display. This is then output to the display.

display_psi() This routine is called from “main()” never to return. The A/D converter routine is called, the pressure is calculated based on the type sensor detected and the pressure applied to the sensor is displayed. The loop then repeats.

sensor_type() This routine determines the type of sensor from reading J1 to J3, setting the full scale pressure for that particular sensor in a variable for use by display_psi().

sensor_slope() This routine determines the slope constant to be used by display_psi() for engineering units output.

main() This is the main routine called from reset. It calls “initio()” to setup the system's I/O. “display_psi()” is called to compute and display the pressure applied to the sensor.

```
                            #pragma option f0;
                            /*

                            THE FOLLOWING 'C' SOURCE CODE IS WRITTEN FOR THE DEVB158 EVALUATION
                            BOARD. IT WAS COMPILED WITH A COMPILER COURTESY OF:

                                                                BYTE CRAFT LTD.
                                                                421 KING ST.
                                                                WATERLOO, ONTARIO
                                                                CANADA  N2J 4E4
                                                                (519)888-6911

                            SOME SOURCE CODE CHANGES MAY BE NECESSARY FOR COMPILATION WITH OTHER
                            COMPILERS.
                                                      BILL LUCAS 2/5/92
                                                      Freescale, SPS

                            Revision history
                            rev. 1.0 initial release 3/19/92
                            rev. 1.1 added additional decimal digit to the MPX2010 sensor. Originally
                            resolved the output to .1 PSI. Modified cvt_bin_dec to output PSI resolved
                            to .01 PSI. WLL 9/25/92

                            */
0800 1700                   #pragma memory ROMPROG  [5888]  @ 0x0800 ;
0050 0096                   #pragma memory RAMPAGE0 [150]   @ 0x0050 ;

                            /*       Vector assignments    */
1FFE                        #pragma vector __RESET   @ 0x1ffe ;
1FFC                        #pragma vector __SWI     @ 0x1ffc ;
1FFA                        #pragma vector IRQ       @ 0x1ffa ;
1FF8                        #pragma vector TIMERCAP  @ 0x1ff8 ;
1FF6                        #pragma vector TIMERCMP  @ 0x1ff6 ;
1FF4                        #pragma vector TIMEROV   @ 0x1ff4 ;
1FF2                        #pragma vector SCI       @ 0x1ff2 ;

                            #pragma has STOP ;
                            #pragma has WAIT ;
                            #pragma has MUL ;

                            /*       Register assignments for the 68HC705B5 microcontroller     */
0000                        #pragma portrw porta   @ 0x00;  /*                                          */
0001                        #pragma portrw portb   @ 0x01;  /*                                          */
0002                        #pragma portrw portc   @ 0x02;  /*                                          */
0003                        #pragma portrw portd   @ 0x03;  /* in  ,-    ,SS   ,SCK  ,MOSI ,MISO,TxD,RxD */
0004                        #pragma portrw ddra    @ 0x04;  /* Data direction, Port A                   */
0005                        #pragma portrw ddrb    @ 0x05;  /* Data direction, Port B                   */
0006                        #pragma portrw ddrc    @ 0x06;  /* Data direction, Port C (all output)      */
0007                        #pragma portrw eeclk   @ 0x07;  /* eeprom/eclk cntl */
0008                        #pragma portrw addata  @ 0x08;  /* a/d data register */
0009                        #pragma portrw adstat  @ 0x09;  /* a/d stat/control */
000A                        #pragma portrw plma    @ 0x0a;  /* pulse length modulation a */
000B                        #pragma portrw plmb    @ 0x0b;  /* pulse length modulation b */
000C                        #pragma portrw misc    @ 0x0c;  /* miscellaneous register */
000D                        #pragma portrw scibaud @ 0x0d;  /* sci baud rate register */
000E                        #pragma portrw scicntl1 @ 0x0e; /* sci control 1  */
000F                        #pragma portrw scicntl2 @ 0x0f; /* sci control 2  */
0010                        #pragma portrw scistat @ 0x10;  /* sci status reg */
0011                        #pragma portrw scidata   @ 0x11;  /* SCI Data  */
0012                        #pragma portrw tcr       @ 0x12;  /* ICIE,OCIE,TOIE,0;0,0,IEGE,OLVL         */
0013                        #pragma portrw tsr       @ 0x13;  /* ICF,OCF,TOF,0; 0,0,0,0                 */
0014                        #pragma portrw icaphi1   @ 0x14;  /* Input Capture Reg (Hi-0x14, Lo-0x15)   */
0015                        #pragma portrw icaplo1   @ 0x15;  /* Input Capture Reg (Hi-0x14, Lo-0x15)   */
0016                        #pragma portrw ocmphi1   @ 0x16;  /* Output Compare Reg (Hi-0x16, Lo-0x17)  */
0017                        #pragma portrw ocmplo1   @ 0x17;  /* Output Compare Reg (Hi-0x16, Lo-0x17)  */
0018                        #pragma portrw tcnthi    @ 0x18;  /* Timer Count Reg (Hi-0x18, Lo-0x19)     */
0019                        #pragma portrw tcntlo    @ 0x19;  /* Timer Count Reg (Hi-0x18, Lo-0x19)     */
001A                        #pragma portrw aregnthi    @ 0x1A;  /* Alternate Count Reg (Hi-$1A, Lo-$1B)   */
001B                        #pragma portrw aregntlo    @ 0x1B;  /* Alternate Count Reg (Hi-$1A, Lo-$1B)   */
001C                        #pragma portrw icaphi2   @ 0x1c;  /* Input Capture Reg (Hi-0x1c, Lo-0x1d)   */
001D                        #pragma portrw icaplo2   @ 0x1d;  /* Input Capture Reg (Hi-0x1c, Lo-0x1d)   */
```

```
001E                            #pragma portrw ocmphi2   @ 0x1e;  /* Output Compare Reg (Hi-0x1e, Lo-0x1f)  */
001F                            #pragma portrw ocmplo2   @ 0x1f;  /* Output Compare Reg (Hi-0x1e, Lo-0x1f)  */

1EFE 74                         #pragma mor @ 0x1efe = 0x74; /* this disables the watchdog counter and does
                                                                not add pull-down resistors on ports B and C */

                                     /* put constants and variables here...they must be global */
                                  /*************************************************************************/

0800 FC 30 DA 7A 36 6E E6 38 FE   const char lcdtab[]={0xfc,0x30,0xda,0x7a,0x36,0x6e,0xe6,0x38,0xfe,0x3e };
0809 3E
                                  /* lcd pattern table  0    1    2    3    4    5    6    7    8    9     */

080A 27 10 03 E8 00 64 00 0A      const long dectable[] = { 10000, 1000, 100, 10 };

0050 0005                         unsigned int digit[5]; /* buffer to hold results from cvt_bin_dec function */

0812 00 96 00 4B 00 96 00 1E 00   const long type[] = {  150,     75,      150,     30,      103   };
081B 67
                                  /*                   MPX2010  MPX2050  MPX2100  MPX2200  MPX2700
                                  The table above will cause the final results of the pressure to
                                  engineering units to display the 1.5, 7.3 and 15.0 devices with a
                                  decimal place in the tens position. The 30 and 103 psi devices will
                                  display in integer units.
                                  */

                                  const long slope_const[]={ 450,418,423,427,432,436,441,445,454,459,
081C 01 C2 01 A2 01 A7 01 AB 01                              463,468,472,477,481,450 };
0825 B0 01 B4 01 B9 01 BD 01 C6
082E 01 CB 01 CF 01 D4 01 D8 01
0837 DD 01 E1 01 C2

0000                              registera areg;    /* processor's A register */

0055                              long atodtemp;     /* temp to accumulate 100 a/d readings for smoothing */

0059                              long slope;        /* multiplier for adc to engineering units conversion */

005B                              int adcnt;         /* a/d converter loop counter */

005C                              long xdcr_offset;  /* initial xdcr offset */

005E                              long sensor_model; /*  installed sensor based on J1..J3 */
0060                              int sensor_index;  /*  determine the location of the decimal pt. */

0061 0063                         unsigned long i,j; /* counter for loops */

0065                              unsigned int k;    /* misc variable */

                                    struct bothbytes
                                       {  int hi;
                                          int lo;
                                       };

                                       union isboth
                                          {  long  l;
                                           struct bothbytes b;
                                          };

0066 0002                                 union isboth q;  /* used for timer set-up */

                                  /*************************************************************************/

                                  /* variables for add32  */
0068 0004                         unsigned long SUM[2];    /*     result        */
006C 0004                         unsigned long ADDEND[2]; /*     one input     */
0070 0004                         unsigned long AUGEND[2]; /*     second input  */

                                  /* variables for sub32  */
0074 0004                         unsigned long MINUE[2];  /*     minuend       */
0078 0004                         unsigned long SUBTRA[2]; /*     subtrahend    */
```

```
007C 0004                       unsigned long DIFF[2];   /*     difference    */

                                /* variables for mul32  */
0080 0004                       unsigned long MULTP[2];  /*     multiplier    */
0084 0004                       unsigned long MTEMP[2];  /*     high order 4 bytes at return */
0088 0004                       unsigned long MULCAN[2]; /*     multiplicand at input, low 4 bytes at return */

                                  /* variables for div32 */
008C 0004                       unsigned long DVDND[2];  /*     Dividend      */
0090 0004                       unsigned long DVSOR[2];  /*     Divisor       */
0094 0004                       unsigned long QUO[2];    /*     Quotient      */
0098                            unsigned int  CNT;       /*     Loop counter  */

                                                    /* The code starts here */

                                  /*************************************************************************/

                                  void add32()
                                  {
                                  #asm

                    *-------------------------------------------------------------------------*
                    * Add two 32-bit values.
                    *    Inputs:
                    *       ADDEND: ADDEND[0..3]  HIGH ORDER BYTE IS ADDEND+0
                    *       AUGEND: AUGEND[0..3]  HIGH ORDER BYTE IS AUGEND+0
                    *    Output:
                    *       SUM: SUM[0..3]  HIGH ORDER BYTE IS SUM+0
                    *-------------------------------------------------------------------------*
                    *
083C B6 6F                        LDA  ADDEND+3   low byte
083E BB 73                        ADD  AUGEND+3
0840 B7 6B                        STA  SUM+3
0842 B6 6E                        LDA  ADDEND+2   medium low byte
0844 B9 72                        ADC  AUGEND+2
0846 B7 6A                        STA  SUM+2
0848 B6 6D                        LDA  ADDEND+1   medium high byte
084A B9 71                        ADC  AUGEND+1
084C B7 69                        STA  SUM+1
084E B6 6C                        LDA  ADDEND     high byte
0850 B9 70                        ADC  AUGEND
0852 B7 68                        STA  SUM
0854 81                           RTS            done
                    *
                                    #endasm
0855 81       RTS                   }

                                  void sub32()
                                  {
                                  #asm
                    *-------------------------------------------------------------------------*
                    * Subtract two 32-bit values.
                    *    Input:
                    *       Minuend: MINUE[0..3]
                    *       Subtrahend: SUBTRA[0..3]
                    *    Output:
                    *       Difference: DIFF[1..0]
                    *-------------------------------------------------------------------------*
                    *
0856 B6 77                        LDA  MINUE+3    low byte
0858 B0 7                         SUB  SUBTRA+3
085A B7 7F                        STA  DIFF+3
085C B6 76                        LDA  MINUE+2    medium low byte
085E B2 7A                        SBC  SUBTRA+2
0860 B7 7E                        STA  DIFF+2
0862 B6 75                        LDA  MINUE+1    medium high byte
0864 B2 79                        SBC  SUBTRA+1
0866 B7 7D                        STA  DIFF+1
0868 B6 74                        LDA  MINUE      high byte
086A B2 78                        SBC  SUBTRA
086C B7 7C                        STA  DIFF
086E 81                           RTS            done
                    *
```

```
                                #endasm
086F 81       RTS               }

                                     void mul32()
                                     {
                                     #asm
                        *-------------------------------------------------------------------------*
                        * Multiply 32-bit value by a 32-bit value
                        *
                        *
                        *    Input:
                        *       Multiplier:    MULTP[0..3]
                        *       Multiplicand:  MULCAN[0..3]
                        *    Output:
                        *       Product:       MTEMP[0..3] AND MULCAN[0..3] MTEMP[0] IS THE HIGH
                        *                          ORDER BYTE AND MULCAN[3] IS THE LOW ORDER BYTE
                        *
                        *   THIS ROUTINE DOES NOT USE THE MUL INSTRUCTION FOR THE SAKE OF USERS NOT
                        *   USING THE HC(7)05 SERIES PROCESSORS.
                        *-------------------------------------------------------------------------*
                        *
                        *
0870 AE 20                        LDX  #32        loop counter
0872 3F 84                        CLR  MTEMP      clean-up for result
0874 3F 85                        CLR  MTEMP+1    *
0876 3F 86                        CLR  MTEMP+2    *
0878 3F 87                        CLR  MTEMP+3    *
087A 36 88                        ROR  MULCAN     low but to carry, the rest one to the right
087C 36 89                        ROR  MULCAN+1   *
087E 36 8A                        ROR  MULCAN+2   *
0880 36 8B                        ROR  MULCAN+3   *
0882 24 18                MNEXT   BCC  ROTATE     if carry is set, do the add
0884 B6 87                        LDA  MTEMP+3    *
0886 BB 83                        ADD  MULTP+3    *
0888 B7 87                        STA  MTEMP+3    *
088A B6 86                        LDA  MTEMP+2    *
088C B9 82                        ADC  MULTP+2    *
088E B7 86                        STA  MTEMP+2    *
0890 B6 85                        LDA  MTEMP+1    *
0892 B9 81                        ADC  MULTP+1    *
0894 B7 85                        STA  MTEMP+1    *
0896 B6 84                        LDA  MTEMP      *
0898 B9 80                        ADC  MULTP      *
089A B7 84                        STA  MTEMP      *
089C 36 84                ROTATE  ROR  MTEMP     else: shift low bit to carry, the rest to the right
089E 36 85                        ROR  MTEMP+1    *
08A0 36 86                        ROR  MTEMP+2    *
08A2 36 87                        ROR  MTEMP+3    *
08A4 36 88                        ROR  MULCAN     *
08A6 36 89                        ROR  MULCAN+1   *
08A8 36 8A                        ROR  MULCAN+2   *
08AA 36 8B                        ROR  MULCAN+3   *
08AC 5A                           DEX            bump the counter down
08AD 26 D3                        BNE  MNEXT     done yet ?
08AF 81                           RTS            done

                                #endasm
08B0 81       RTS               }

                                  void div32()
                                  {
                                  #asm
                         *
                         *-------------------------------------------------------------------------*
                         * Divide 32 bit by 32 bit unsigned integer routine
                         *
                         *    Input:
                         *       Dividend:  DVDND [+0..+3] HIGH ORDER BYTE IS DVND+0
                         *       Divisor:   DVSOR [+0..+3] HIGH ORDER BYTE IS DVSOR+0
                         *    Output:
                         *       Quotient:  QUO [+0..+3]   HIGH ORDER BYTE IS QUO+0
                         *-------------------------------------------------------------------------*
                         *
```

```
08B1 3F 94                      CLR  QUOzero result registers
08B3 3F 95                      CLR  QUO+1      *
08B5 3F 96                      CLR  QUO+2      *
08B7 3F 97                      CLR  QUO+3      *
08B9 A6 01                      LDA  #1         initial loop count
08BB 3D 90                      TST  DVSOR      if the high order bit is set..no need to shift DVSOR
08BD 2B 0F                      BMI  DIV153
                     *
08BF 4C                DIV151  INCA            bump the loop counter
08C0 38 93              ASL  DVSOR+3    now shift the divisor until the high order bit = 1
08C2 39 92                      ROL  DVSOR+2
08C4 39 91                      ROL  DVSOR+1    *
08C6 39 90                      ROL  DVSOR      *
08C8 2B 04              BMI  DIV153     done if high order bit = 1
08CA A1 21                      CMP  #33        have we shifted all possible bits in the DVSOR yet ?
08CC 26 F1                      BNE  DIV151     no
                     *
08CE B7 9              DIV153   STA  CNT        save the loop counter so we can do the divide
                     *
08D0 B6 8F             DIV163   LDA  DVDND+3    sub 32 bit divisor from dividend
08D2 B0 93                      SUB  DVSOR+3    *
08D4 B7 8F                      STA  DVDND+3    *
08D6 B6 8E                      LDA  DVDND+2    *
08D8 B2 92                      SBC  DVSOR+2    *
08DA B7 8E                      STA  DVDND+2    *
08DC B6 8D                      LDA  DVDND+1    *
08DE B2 91                      SBC  DVSOR+1    *
08E0 B7 8D                      STA  DVDND+1    *
08E2 B6 8C                      LDA  DVDND      *
08E4 B2 90                      SBC  DVSOR      *
08E6 B7 8C                      STA  DVDND      *
08E8 24 1B                      BCC  DIV165     carry is clear if DVSOR was larger than DVDND
                     *
08EA B6 8F                      LDA  DVDND+3    add the divisor back...was larger than the dividend
08EC BB 93                      ADD  DVSOR+3    *
08EE B7 8F                      STA  DVDND+3    *
08F0 B6 8E                      LDA  DVDND+2    *
08F2 B9 92                      ADC  DVSOR+2    *
08F4 B7 8E                      STA  DVDND+2    *
08F6 B6 8D                      LDA  DVDND+1    *
08F8 B9 91                      ADC  DVSOR+1    *
08FA B7 8D                      STA  DVDND+1    *
08FC B6 8C                      LDA  DVDND      *
08FE B9 90                      ADC  DVSOR      *
0900 B7 8C                      STA  DVDND      *
0902 98                         CLC             this will clear the respective bit in QUO due to
                     *                            the need to add DVSOR back to DVND
0903 20 01                      BRA  DIV167
0905 99                DIV165   SEC             this will set the respective bit in QUO
0906 39 97             DIV167   ROL  QUO+3      set or clear the low order bit in QUO based on above
0908 39 96                      ROL  QUO+2      *
090A 39 95                      ROL  QUO+1      *
090C 39 94                      ROL  QUO        *
090E 34 90                      LSR  DVSOR     divide the divisor by 2
0910 36 91                      ROR  DVSOR+1    *
0912 36 92                      ROR  DVSOR+2    *
0914 36 93                      ROR  DVSOR+3    *
0916 3A 98                      DEC  CNT        bump the loop counter down
0918 26 B6                      BNE  DIV163     finished yet ?
091A 81                         RTSyes
                     *

                                #endasm
091B 81        RTS              }

                                /*****************************************************************************/

                                /* These interrupts are not used...give them a graceful return for
                                   some reason one occurs */

1FFC 09 1C                      __SWI(){}
091C 80        RTI
1FFA 09 1D                      IRQ(){}
```

```
091D 80       RTI
1FF8 09 1E                    TIMERCAP(){}
091E 80       RTI
1FF4 09 1F                    TIMEROV(){}
091F 80       RTI
1FF2 09 20                    SCI(){}
0920 80       RTI

                                /*************************************************************************/

                                void sensor_type()
                                {
0921 B6 03    LDA    $03      k = portd & 0x0e;  /* we only care about bits 1..3 */
0923 A4 0E    AND    #$0E
0925 B7 65    STA    $65
0927 34 65    LSR    $65      k = k >> 1;    /* right justify the variable */
0929 B6 65    LDA    $65       if ( k > 4 )
092B A1 04    CMP    #$04
092D 23 0C    BLS    $093B
                                   { /* we have a set-up error in wire jumpers J1 - J3 */
092F 3F 02    CLR    $02         portc = 0;     /*   */
0931 A6 6E    LDA    #$6E        portb = 0x6e;  /* S */
0933 B7 01    STA    $01
0935 A6 CE    LDA    #$CE        porta = 0xce;  /* E */
0937 B7 00    STA    $00
0939 20 FE    BRA    $0939          while(1);
                                     }
093B B6 65    LDA    $65      sensor_index = k;
093D B7 60    STA    $60
093F 97       TAX             sensor_model = type[k];
0940 58       LSLX
0941 D6 08 12 LDA    $0812,X
0944 B7 5E    STA    $5E
0946 D6 08 13 LDA    $0813,X
0949 B7 5F    STA    $5F
094B 81       RTS             }

                                /*************************************************************************/

                                void sensor_slope()
                                {
094C B6 03    LDA    $03      k=portd & 0xf0;  /* we only care about bits 4..7 */
094E A4 F0    AND    #$F0
0950 B7 65    STA    $65
0952 34 65    LSR    $65        k = k >> 4;    /* right justify the variable */
0954 34 65    LSR    $65
0956 34 65    LSR    $65
0958 34 65    LSR    $65
095A BE 65    LDX    $65      slope = slope_const[k];
095C 58       LSLX
095D D6 08 1C LDA    $081C,X
0960 B7 59    STA    $59
0962 D6 08 1D LDA    $081D,X
0965 B7 5A    STA    $5A
0967 81       RTS             }

                                /*************************************************************************/

                                void delay(void)  /* just hang around for a while */
                                {
0968 3F 62    CLR    $62      for (i=0; i<20000; ++i);
096A 3F 61    CLR    $61
096C B6 62    LDA    $62
096E A0 20    SUB    #$20
0970 B6 61    LDA    $61
0972 A2 4E    SBC    #$4E
0974 24 08    BCC    $097E
0976 3C 62    INC    $62
0978 26 0     BNE    $097C
097A 3C 61    IN     $61
097C 20 EE    BRA    $096C
097E 81       RTS             }
```

```
                                /*****************************************************************************/

                                read_a2d(void)
                                {
                                /* read the a/d converter on channel 5 and accumulate the result
                                in atodtemp */

097F 3F 56     CLR    $56          atodtemp=0;   /* zero for accumulation */
0981 3F 55     CLR    $55
0983 3F 5B     CLR    $5B            for ( adcnt = 0 ; adcnt<100; ++adcnt) /* do 100 a/d conversions */
0985 B6 5B     LDA    $5B
0987 A8 80     EOR    #$80
0989 A1 E4     CMP    #$E4
098B 24 21     BCC    $09AE
                                        {
098D A6 20     LDA    #$20           adstat = 0x20;  /* convert on channel 0 */
098F B7 09      STA    $09
0991 0F 09 FD BRCLR 7,$09,$0991        while (!(adstat & 0x80));  /* wait for a/d to complete */
0994 B6 08     LDA    $08              atodtemp = addata + atodtemp;
0996 3F 57     CLR    $57
0998 B7 58     STA    $58
099A BB 56     ADD    $56
099C B7 58     STA    $58
099E B6 57     LDA    $57
09A0 B9 55     ADC    $55
09A2 B7 57     STA    $57
09A4 B7 55     STA    $55
09A6 B6 58     LDA    $58
09A8 B7 56     STA    $56
                                      }
09AA 3C 5B     INC    $5B
09AC 20 D7     BRA    $0985
09AE B6 56     LDA    $56            atodtemp = atodtemp/100;
09B0 B7 58     STA    $58
09B2 B6 55     LDA    $55
09B4 B7 57     STA    $57
09B6 3F 9A     CLR    $9A
09B8 A6 64     LDA    #$64
09BA B7 9B     STA    $9B
09BC CD 0B F1  JSR    $0BF1
09BF CD 0C 22  JSR    $0C22
09C2 BF 55     STX    $55
09C4 B7 56     STA    $56
09C6 81        RTS                   return atodtemp;
                                }

                                /*****************************************************************************/

                                void fixcompare (void)  /* sets-up the timer compare for the next interrupt */
                                   {
09C7 B6 18     LDA    $18            q.b.hi =tcnthi;
09C9 B7 66     STA    $66
09CB B6 19     LDA    $19               q.b.lo = tcntlo;
09CD B7 67     STA    $67
09CF AB 4C     ADD    #$4C               q.l +=7500;  /* ((4mhz xtal/2)/4) = counter period = 2us.*7500 = 15ms. */
09D1 B7 67     STA    $67
09D3 B6 66     LDA    $66
09D5 A9 1D     ADC    #$1D
09D7 B7 66     STA    $66
09D9 B7 16     STA    $16                ocmphi1 = q.b.hi;
09DB B6 13     LDA    $13               areg=tsr;  /* dummy read */
09DD B6 67     LDA    $67            ocmplo1 = q.b.lo;
09DF B7 17     STA    $17
09E1 81        RTS                   }

                                /*****************************************************************************/

                                void TIMERCMP (void)    /* timer service module */
1FF6 09 E2                      {
09E2 33 02     COM    $02             portc =~ portc;    /* service the lcd by inverting the ports */
09E4 33 01     COM    $01             portb =~ portb;
09E6 33 00     COM    $00             porta =~ porta;
```

```
09E8 AD DD     BSR    $09C7          fixcompare();
09EA 80        RTI                   }

                                     /*************************************************************************/

                                     void adzero(void)  /* called by initio() to save initial xdcr's zero
                                                           pressure offset voltage output */
                                     {

09EB 3F 64     CLR    $64            for ( j=0; j<20; ++j)  /* give the sensor time to "warm-up" and the
09ED 3F 63     CLR    $63
09EF B6 64     LDA    $64
09F1 A0 14     UB     #$14
09F3 B6 63     LDA    $63
09F5 A2 00     SBC    #$00
09F7 24 0B     BCC    $0A04
                                                              power supply time to settle down */
                                     {
09F9 CD 09 68  JSR    $0968             delay();
                                     }
09FC 3C 64     INC    $64
09FE 26 02     BNE    $0A02
0A00 3C 63     INC    $63
0A02 20 EB     BRA    $09EF
0A04 CD 09 7F  JSR    $097F             xdcr_offset =  read_a2d();
0A07 3F 5C     CLR    $5C
0A09 B7 5D     STA    $5D
0A0B 81        RTS                 }

                                     /*************************************************************************/

                                     void initio (void)     /*  setup the I/O */
                                        {
0A0C A6 20     LDA    #$20              adstat = 0x20; /* power-up the A/D */
0A0E B7 09     STA    $09
0A10 3F 02     CLR    $02               porta = portb = portc = 0;
0A12 3F 01     CLR    $01
0A14 3F 00     CLR    $00
0A16 A6 FF     LDA    #$FF              ddra = ddrb = ddrc = 0xff;
0A18 B7 06     STA    $06
0A1A B7 05     STA    $05
0A1C B7 04     STA    $04
0A1E B6 13     LDA    $13               areg=tsr;  /* dummy read */
0A20 3F 1E     CLR    $1E               ocmphi1 = ocmphi2 = 0;
0A22 3F 16     CLR    $16
0A24 B6 1F     LDA    $1F               areg = ocmplo2; /* clear out output compare 2 if it happens to be set */
0A26 AD 9F     BSR    $09C7             fixcompare(); /* set-up for the first timer interrupt */
0A28 A6 40     LDA    #$40              tcr = 0x40;
0A2A B7 12     STA    $12
0A2C 9A        CLI                      CLI;  /* let the interrupts begin !  */
                                     /* write CAL to the display */
0A2D A6 CC     LDA    #$CC              portc = 0xcc; /* C */
0A2F B7 02     STA    $02
0A31 A6 BE     LDA    #$BE              portb = 0xbe; /* A */
0A33 B7 01     STA    $01
0A35 A6 C4     LDA    #$C4              porta = 0xc4; /* L */
0A37 B7 00     STA    $00
0A39 CD 09 21  JSR    $0921             sensor_type(); /* get the model of the sensor based on J1..J3 */
0A3C AD AD     BSR    $09EB             adzero();  /* auto zero */
0A3E 81        RTS                   }

                                     /*************************************************************************/

                                     void cvt_bin_dec(unsigned long arg)

                                     /* First converts the argument to a five digit decimal value. The msd is in
                                     the lowest address. Then leading zero suppress the value and write it to the
                                     display ports.
                                        The argument value is 0..65535 decimal. */

009D                               {
0A3F BF 9D     STX    $9D
0A41 B7 9E     STA    $9E
```

```
009F                            char i;
00A0                            unsigned long l;
0A43 3F 9F     CLR    $9F         for ( i=0; i < 5; ++i )
0A45 B6 9F     LDA    $9F
0A47 A1 05     CMP    #$05
0A49 24 07     BCC    $0A52

                                    {
0A4B 97        TAX                  digit[i] = 0x0;  /* put blanks in all digit positions */
0A4C 6F 50     CL     $50,X
                                    }
0A4E 3C 9F     INC    $9F
0A50 20 F3     BRA    $0A45
0A52 3F 9F     CLR    $9F           for ( i=0; i < 4; ++i )
0A54 B6 9F     LDA    $9F
0A56 A1 04     CMP    #$04
0A58 24 7A     BCC    $0AD4

                                       {
0A5A 97        TAX                     if ( arg  >= dectable [i] )
0A5B 58        LSLX
0A5C D6 08 0B  LDA    $080B,X
0A5F B0 9E     SUB    $9E
0A61 B7 58     STA    $58
0A63 B6 9D     LDA    $9D
0A65 A8 80     EOR    #$80
0A67 B7 57     STA    $57
0A69 D6 08 0A  LDA    $080A,X
0A6C A8 80     EOR    #$80
0A6E B2 57     SBC    $57

0A70 BA 58      ORA    $58
0A72 22 5C      BHI    $0AD0

                                          {
0A74 BE 9F      LDX    $9F                  l = dectable[i];
0A76 58         LSLX
0A77 D6 08 0A   LDA    $080A,X
0A7A B7 A0      STA    $A0
0A7C D6 08 0B   LDA    $080B,X
0A7F B7 A1      STA    $A1
0A81 B6 9E      LDA    $9E                  digit[i] = arg / l;
0A83 B7 58      STA    $58
0A85 B6 9D      LDA    $9D
0A87 B7 57      STA    $57
0A89 B6 A0      LDA    $A0
0A8B B7 9A      STA    $9A
0A8D B6 A1      LDA    $A1
0A8F B7 9B      STA    $9B
0A91 CD 0B F1   JSR    $0BF1
0A94 CD 0C 22   JSR    $0C22
0A97 BF 57      STX    $57
0A99 B7 58      STA    $58
0A9B BE 9F      LDX    $9F
0A9D E7 50      STA    $50,X
0A9F BE 9F      LDX    $9F                  arg = arg-(digit[i] * l);
0AA1 E6 50      LDA    $50,X
0AA3 3F 57      CLR    $57
0AA5 B7 58      STA    $58
0AA7 B6 A0      LDA    $A0
0AA9 B7 9A      STA    $9A
0AAB B6 A1      LDA    $A1
0AAD B7 9B      STA    $9B
0AAF CD 0B D2   JSR    $0BD2
0AB2 BF 57      STX    $57
0AB4 B7 58      STA    $58
0AB6 33 57      COM    $57
0AB8 30 58      NEG    $58
0ABA 26 02      BNE    $0ABE
0ABC 3C 57      INC    $57
0ABE B6 58      LDA    $58
0AC0 BB 9E      ADD    $9E
0AC2 B7 58      STA    $58
0AC4 B6 57      LDA    $57
0AC6 B9 9D      ADC    $9D
0AC8 B7 57      STA    $57
0ACA B7 9D      STA    $9D
```

```
0ACC B6 58     LDA    $58
0ACE B7 9E     STA    $9E
                                                }
                                              }
0AD0 3C 9F     INC    $9F
0AD2 20 80     BRA    $0A54
0AD4 B6 9E     LDA    $9E          digit[i] = arg;
0AD6 B7 58     STA    $58
0AD8 B6 9D     LDA    $9D
0ADA B7 57     STA    $57
0ADC BE 9F     LDX    $9F
0ADE B6 58     LDA    $58
0AE0 E7 50     STA    $50,X

                                    /* now zero suppress and send the lcd pattern to the display */
0AE2 9B        SEI                 SEI;
0AE3 3D 52     TST    $52          if ( digit[2] == 0 )   /* leading zero suppression */
0AE5 26 04     BNE    $0AEB
0AE7 3F 02     CLR    $02            portc = 0;
0AE9 20 07     BRA    $0AF2            else
0AEB BE 52     LDX    $52               portc = ( lcdtab[digit[2]] );     /* 100's digit */
0AED D6 08 00  LDA    $0800,X
0AF0 B7 02     STA    $02
0AF2 3D 52     TST    $52                if ( digit[2] == 0 && digit[3] == 0 )
0AF4 26 08     BNE    $0AFE
0AF6 3D 53     TST    $53
0AF8 26 04     BNE    $0AFE
0AFA 3F 01     CLR    $01                  portb=0;
0AFC 20 07     BRA    $0B05                  else
0AFE BE 53     LDX    $53                    portb = ( lcdtab[digit[3]] );    /* 10's digit */
0B00 D6 08 00  LDA    $0800,X

0B03 B7 01     STA    $01
0B05 BE 54     LDX    $54                      porta = ( lcdtab[digit[4]] );   /* 1's digit */
0B07 D6 08 00  LDA    $0800,X
0B0A B7 00     STA    $00
                                    /* place the decimal point only if the sensor is 15 psi or 7.5 psi */
0B0C B6 60     LDA    $60            if ( sensor_index < 3 )
0B0E A8 80     EOR    #$80
0B10 A1 83     CMP    #$83
0B12 24 08     BCC    $0B1C
0B14 BE 54     LDX    $54              porta = ( lcdtab[digit[4]]+1 ); /* add the decimal point to the lsd */
0B16 D6 08 00  LDA    $0800,X
0B19 4C        INCA
0B1A B7 00     STA    $00
0B1C 3D 60     TST    $60               if(sensor_index ==0) /* special case */
0B1E 26 0F     BNE    $0B2F
                                           {
0B20 BE 54     LDX    $54                 porta = ( lcdtab[digit[4]] ); /* get rid of the decimal at lsd */
0B22 D6 08 00  LDA    $0800,X
0B25 B7 00     STA    $00
0B27 BE 53     LDX    $53                 portb = ( lcdtab[digit[3]]+1 ); /* decimal point at middle digit */
0B29 D6 08 00  LDA    $0800,X
0B2C 4C        INCA
0B2D B7 01     STA    $01
                                         }
0B2F 9A        CLI                 CLI;
0B30 CD 09 68  JSR    $0968                delay();
0B33 81        RTS                 }

                                    /*****************************************************************/

                                    void display_psi(void)
                                    /*
                                      At power-up it is assumed that the pressure or vacuum port of
                                      the sensor is open to atmosphere. The code in initio() delays
                                      for the sensor and power supply to stabilize. One hundred A/D
                                      conversions are averaged. That result is called xdcr_offset.
                                      This routine calls the A/D routine which performs one hundred
                                      conversions, divides the result by 100 and returns the value.
                                      If the value returned is less than or equal to the xdcr_offset,
                                      the value of xdcr_offset is substituted. If the value returned
                                      is greater than xdcr_offset, xdcr_offset is subtracted from the
```

```
                                    returned value.
                                  */

                                 {
                                   while(1)
                                      {
0B34 CD 09 7F  JSR    $097F          atodtemp = read_a2d();  /* atodtemp = raw a/d ( 0..255 ) */
0B37 3F 55     CLR    $55
0B39 B7 56     STA    $56
0B3B B0 5D     SUB    $5D              if ( atodtemp <= xdcr_offset )
0B3D B7 58     STA    $58
0B3F B6 5C     LDA    $5C
0B41 A8 80     EOR    #$80
0B43 B7 57     STA    $57
0B45 B6 55     LDA    $55
0B47 A8 80     EOR    #$80
0B49 B2 57     SBC    $57
0B4B BA 58     ORA    $58
0B4D 22 08     BHI    $0B57
0B4F B6 5C     LDA    $5C                atodtemp = xdcr_offset;
0B51 B7 55     STA    $55
0B53 B6 5D     LDA    $5D
0B55 B7 56     STA    $56
0B57 B6 56     LDA    $56              atodtemp -=  xdcr_offset; /* remove the offset */
0B59 B0 5D     SUB    $5D
0B5B B7 56     STA    $56
0B5D B6 55     LDA    $55
0B5F B2 5C     SBC    $5C
0B61 B7 55     STA    $55
0B63 CD 09 4C  JSR    $094C             sensor_slope();  /* establish the slope constant for this output */
0B66 B6 56     LDA    $56              atodtemp *= sensor_model;
0B68 B7 58     STA    $58
0B6A B6 55     LDA    $55
0B6C B7 57     STA    $57
0B6E B6 5E     LDA    $5E

0B70 B7 9A     STA    $9A
0B72 B6 5F     LDA    $5F
0B74 B7 9B     STA    $9B
0B76 CD 0B D2  JSR    $0BD2
0B79 BF 55     STX    $55
0B7B B7 56     STA    $56
0B7D 3F 89     CLR    $89               MULTP[0] = MULCAN[0] = 0;
0B7F 3F 88     CLR    $88
0B81 3F 81     CLR    $81
0B83 3F 80     CLR    $80
0B85 9F        TXA                      MULTP[1] = atodtemp;
0B86 B7 82     STA    $82
0B88 B6 56     LDA    $56
0B8A B7 83     STA    $83
0B8C B6 59     LDA    $59                MULCAN[1] = slope;
0B8E B7 8A     STA    $8A
0B90 B6 5A     LDA    $5A
0B92 B7 8B     STA    $8B
0B94 CD 08 70  JSR    $0870              mul32();   /* analog value * slope based on J1 through J3 */
0B97 3F 90     CLR    $90                DVSOR[0] = 1;   /*  now divide by 100000 */
0B99 A6 01     LDA    #$01
0B9B B7 91     STA    $91
0B9D A6 86     LDA    #$86                 DVSOR[1] = 0x86a0;
0B9F B7 92     STA    $92
0BA1 A6 A0     LDA    #$A0
0BA3 B7 93     STA    $93
0BA5 B6 88     LDA    $88                    DVDND[0] = MULCAN[0];
0BA7 B7 8C     STA    $8C
0BA9 B6 89     LDA    $89
0BAB B7 8D     STA    $8D
0BAD B6 8A     LDA    $8A                    DVDND[1] = MULCAN[1];
0BAF B7 8E     STA    $8E
0BB1 B6 8B     LDA    $8B
0BB3 B7 8F     STA    $8F
0BB5 CD 08 B1  JSR    $08B1                   div32();
0BB8 B6 96     LDA    $96                    atodtemp = QUO[1]; /* convert to psi */
0BBA B7 55     STA    $55
0BBC B6 97     LDA    $97
```

```
0BBE B7 56     STA    $56
0BC0 BE 55     LDX    $55              cvt_bin_dec( atodtemp ); /* convert to decimal and display */
0BC2 CD 0A 3F  JSR    $0A3F
0BC5 CC 0B 34  JMP    $0B34          }
0BC8 81        RTS                 }

                                     /*************************************************************************/

                                     void main()
                                     {
0BC9 CD 0A 0C  JSR    $0A0C         initio(); /* set-up the processor's i/o */
0BCC CD 0B 34  JSR    $0B34          display_psi();
0BCF 20 FE     BRA    $0BCF           while(1);   /* should never get back to here */
0BD1 81        RTS                 }
0BD2 BE 58     LDX    $58
0BD4 B6 9B     LDA    $9B
0BD6 42        MUL
0BD7 B7 A4     STA    $A4
0BD9 BF A5     STX    $A5
0BDB BE 57     LDX    $57
0BDD B6 9B     LDA    $9B
0BDF 42        MUL
0BE0 BB A5     ADD    $A5
0BE2 B7 A5     STA    $A5
0BE4 BE 58     LDX    $58
0BE6 B6 9A     LDA    $9A
0BE8 42        MUL
0BE9 BB A5     ADD    $A5
0BEB B7 A5     STA    $A5
0BED 97        TAX
0BEE B6 A4     LDA    $A4
0BF0 81        RTS
0BF1 3F A4     CLR    $A4
0BF3 5F        CLRX
0BF4 3F A2     CLR    $A2
0BF6 3F A3     CLR    $A3
0BF8 5C        INCX
0BF9 38 58     LSL    $58


0BFB 39 57     ROL    $57
0BFD 39 A2     ROL    $A2
0BFF 39 A3     ROL    $A3
0C01 B6 A2     LDA    $A2
0C03 B0 9B     SUB    $9B
0C05 B7 A2     STA    $A2
0C07 B6 A3     LDA    $A3
0C09 B2 9A     SBC    $9A
0C0B B7 A3     STA    $A3
0C0D 24 0D     BCC    $0C1C
0C0F B6 9B     LDA    $9B
0C11 BB A2     ADD    $A2
0C13 B7 A2     STA    $A2
0C15 B6 9A     LDA    $9A
0C17 B9 A3     ADC    $A3
0C19 B7 A3     STA    $A3
0C1B 99        SEC
0C1C 59        ROLX
0C1D 39 A4     ROL    $A4
0C1F 24 D8     BCC    $0BF9
0C21 81        RTS
0C22 53        COMX
0C23 9F        TXA
0C24 BE A4     LDX    $A4
0C26 53        COMX
0C27 81        RTS
1FFE 0B C9

SYMBOL TABLE

LABEL        VALUE  LABEL        VALUE  LABEL        VALUE  LABEL        VALUE
```

```
ADDEND       006C | AUGEND       0070 | CNT          0098 | DIFF         007C
DIV151       08BF | DIV153       08CE | DIV163       08D0 | DIV165       0905
DIV167       0906 | DVDND        008C | DVSOR        0090 | IRQ          091D
MINUE        0074 | MNEXT        0882 | MTEMP        0084 | MULCAN       0088
MULTP        0080 | QUO          0094 | ROTATE       089C | SCI          0920
SUBTRA       0078 | SUM          0068 | TIMERCAP     091E | TIMERCMP     09E2
TIMEROV      091F | __LDIV       0BF1 | __LongIX     009A | __MAIN       0BC9
__MUL        0000 | __MUL16x16   0BD2 | __RDIV       0C22 | __RESET      1FFE
__STARTUP    0000 | __STOP       0000 | __SWI        091C | __WAIT       0000
__longAC     0057 | adcnt        005B | add32        083C | addata       0008
adstat       0009 | adzero       09EB | aregnthi     001A | aregntlo     001B
arg          009D | atodtemp     0055 | b            0000 | bothbytes    0002
cvt_bin_dec  0A3F | ddra         0004 | ddrb         0005 | ddrc         0006
dectable     080A | delay        0968 | digit        0050 | display_psi  0B34
div32        08B1 | eeclk        0007 | fixcompare   09C7 | hi           0000
i            0061 | icaphi1      0014 | icaphi2      001C | icaplo1      0015
icaplo2      001D | initio       0A0C | isboth       0002 | j            0063
k            0065 | l            0000 | lcdtab       0800 | lo           0001
main         0BC9 | misc         000C | mul32        0870 | ocmphi1      0016
ocmphi2      001E | ocmplo1      0017 | ocmplo2      001F | plma         000A
plmb         000B | porta        0000 | portb        0001 | portc        0002
portd        0003 | q            0066 | read_a2d     097F | scibaud      000D
scicntl1     000E | scicntl2     000F | scidata      0011 | scistat      0010
sensor_index 0060 | sensor_model 005E | sensor_slope 094C | sensor_type  0921
slope        0059 | slope_const  081C | sub32        0856 | tcnthi       0018
tcntlo       0019 | tcr          0012 | tsr          0013 | type         0812
xdcr_offset  005C |

MEMORY USAGE MAP ('X' = Used, '-' = Unused)

0800 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0840 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0880 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
08C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX

0900 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0940 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0980 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
09C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX

0A00 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0A40 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0A80 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0AC0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX

0B00 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0B40 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0B80 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0BC0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX

0C00 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXX-------- ----------------
0C40 : ---------------- ---------------- ---------------- ----------------
0C80 : ---------------- ---------------- ---------------- ----------------
0CC0 : ---------------- ---------------- ---------------- ----------------

1E00 : ---------------- ---------------- ---------------- ----------------
1E40 : ---------------- ---------------- ---------------- ----------------
1E80 : ---------------- ---------------- ---------------- ----------------
1EC0 : ---------------- ---------------- ---------------- --------------X-

1F00 : ---------------- ---------------- ---------------- ----------------
1F40 : ---------------- ---------------- ---------------- ----------------
1F80 : ---------------- ---------------- ---------------- ----------------
1FC0 : ---------------- ---------------- ---------------- --XXXXXXXXXXXXXX

All other memory blocks unused.

Errors             :    0
Warnings           :    0
```

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

AN1315
Rev. 2
05/2005