

AN1305

An Evaluation System for Direct Interface of the MPX5100 Pressure Sensor with a Microprocessor

Prepared by: Bill Lucas
Discrete Applications Engineering

INTRODUCTION

Interfacing pressure sensors to analog-to-digital converters or microprocessors with on-chip A/D converters has been a challenge that most engineers do not enjoy accepting. Recent design advances in pressure sensing technology have allowed the engineer to directly interface a pressure sensor to an A/D converter with no additional active

components. This has been made possible by integrating a temperature compensated pressure sensor element and active linear circuitry on the same die. A description of an evaluation board that shows the ease of interfacing a signal conditioned pressure sensor to an A/D converter is presented here.

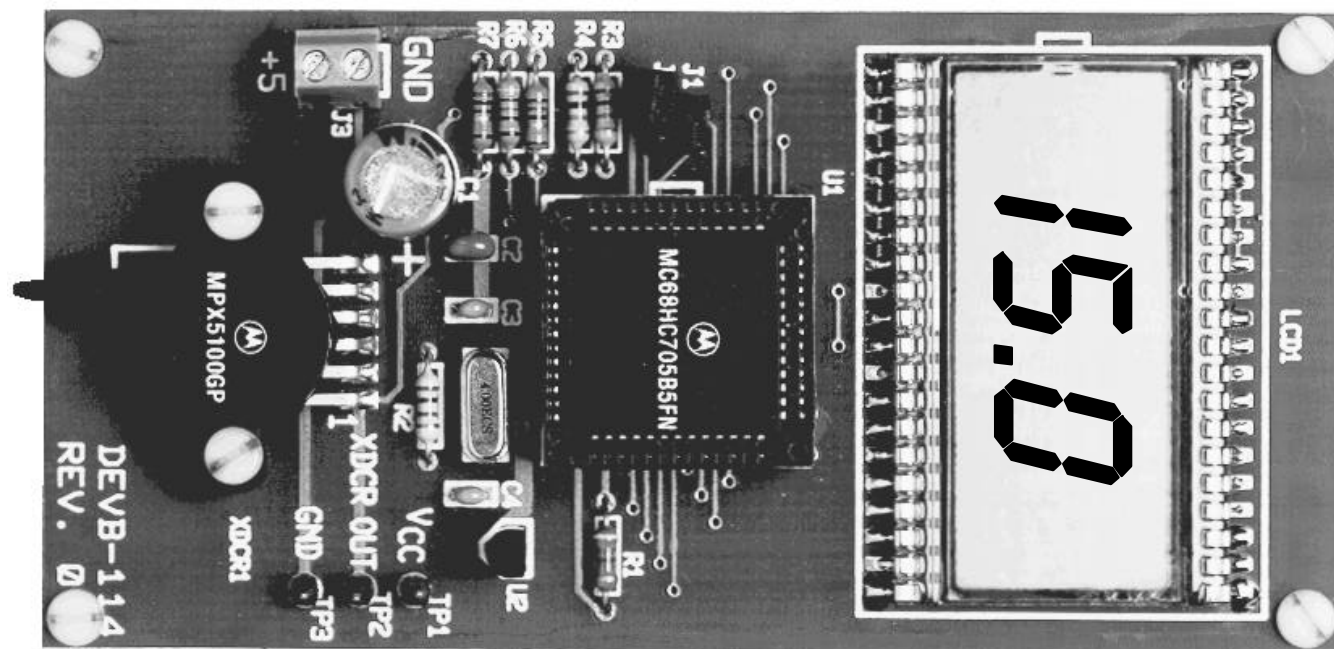


Figure 1. DEVB-114 MPX5100 Evaluation Module

PURPOSE

This evaluation system, shown in Figure 1, demonstrates the ease of operation and interfacing of the Motorola MPX5100 series pressure sensors with on-chip temperature compensation, calibration and amplification. The board may be used to evaluate the sensor's suitability for a specific application.

DESCRIPTION

The DEVB-114 evaluation board is constructed on a small printed circuit board. It is powered from a single +5 Vdc regulated power supply. The system will display the pressure applied to the MPX5100 sensor in pounds per square inch. The range is 0 PSI through 15 PSI, resolved to 0.1 PSI. No potentiometers are used in the system to adjust the span and

offset. The sensor's zero 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 MPX5100 is repeatable from unit to unit. There is a facility for a small "rubbering" of the slope constant built into the program. It is accomplished with jumpers J1 and J2, and is explained in the Operation section. The board contents are further described in the schematic, silk screen plot, and parts list that appear in Figures 2, 3 and Table 1.

BASIC CIRCUIT

The evaluation board consists of three basic subsystems: an MPX5100GP pressure sensor, a four digit liquid crystal display (only three digits and a decimal are used) and a programmed microprocessor with the necessary external circuitry to support the operation of the microprocessor.

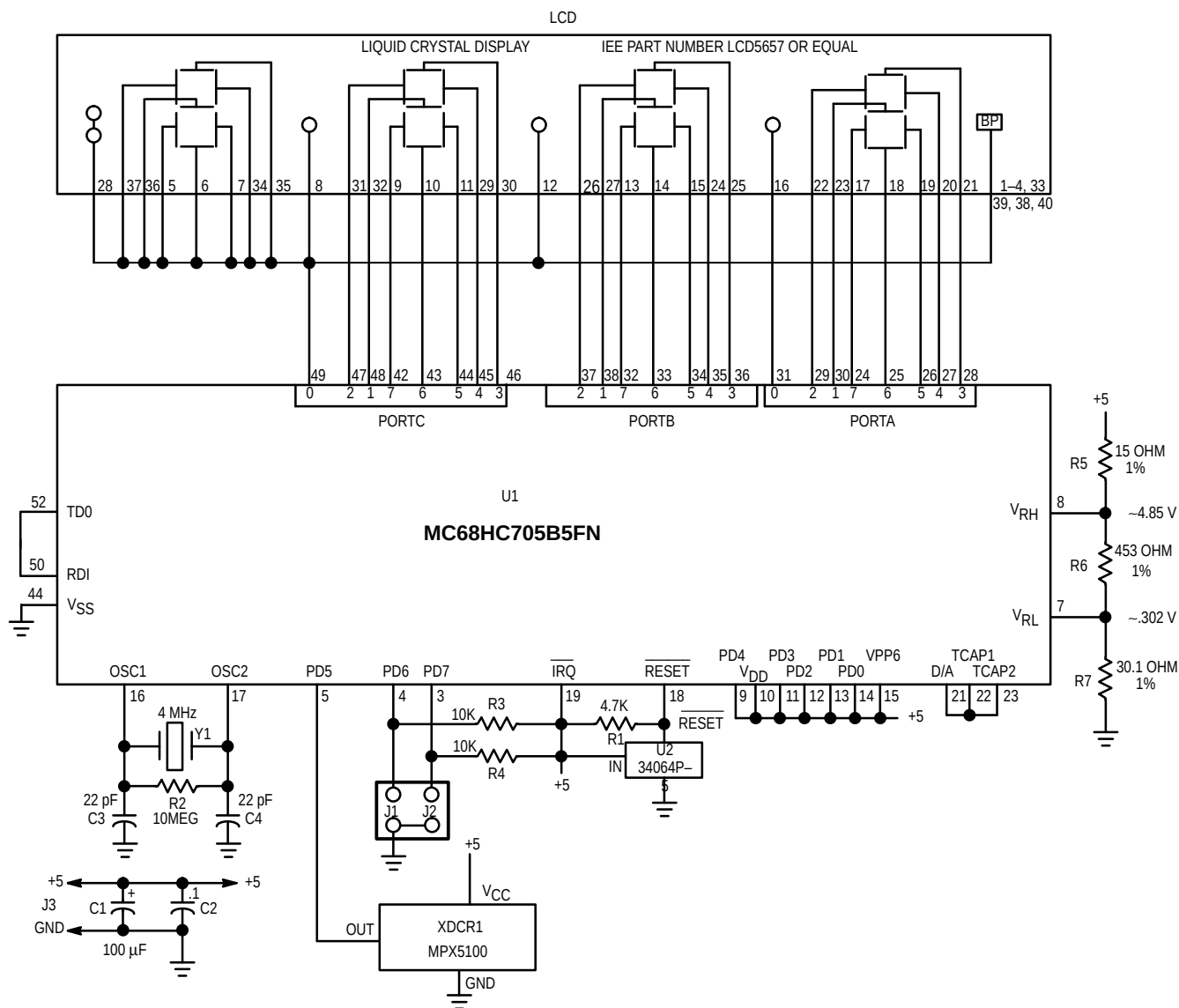


Figure 2. DEVB-114 System Schematic

Table 1. DEVB-114 Parts List

Designators	Quant.	Description	Rating	Manufacturer	Part Number
C1	1	100 μ F Electrolytic Capacitor	25 Vdc	Sprague	513D107M025BB4
C2	1	0.1 μ F Ceramic Capacitor	50 Vdc	Sprague	1C105Z5U104M050B
C3, C4	2	22 pF Ceramic Capacitor	100 Vdc	Mepco/Centralab	CN15A220K
J1, J2	1	Dual Row Straight .025 Pins Arranged On .1" Grid		Molex	10-89-1043
LCD	1	Liquid Crystal Display		AMPEREX	LTD226R-12
R1	1	4.7 k Ohm Resistor			
R2	1	10 Meg Ohm Resistor			
R3, R4	2	10 k Ohm Resistor			
R5	1	15 Ohm 1% 1/4 W Resistor			
R6	1	453 Ohm 1% 1/4 W Resistor			
R7	1	30.1 Ohm 1% 1/4 W Resistor			
XDCR1	1	Pressure Sensor		Motorola	MPX5100GP
U1	1	Microprocessor		Motorola Motorola	MC68HC705B5FN or XC68HC705B5FN
U2	1	Under Voltage Detector		Motorola	MC34064P-5
Y1	1	Crystal (Low Profile)	4.0 MHz	ECS	ECS-40-S-4
No Designator	1	52 Pin PLCC Socket		AMP	821-575-1
No Designator	2	Jumpers For J1 and J2		Molex	15-29-1025
No Designator	1	Bare Printed Circuit Board			

Note: 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.

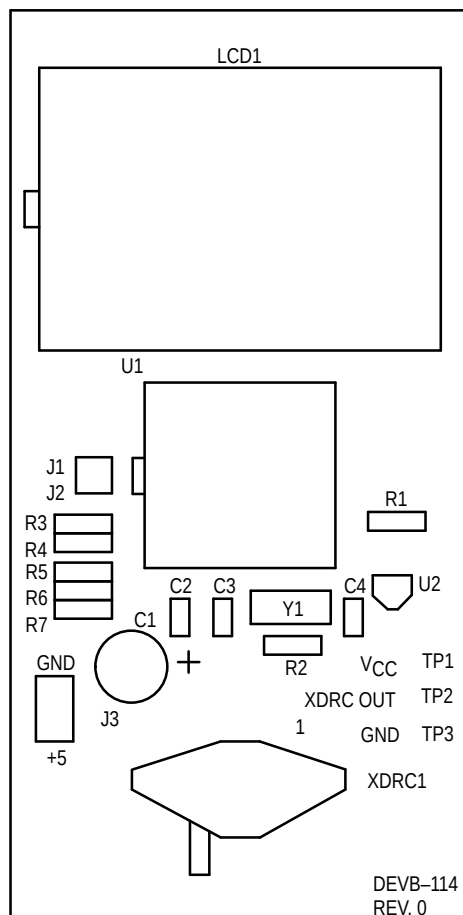


Figure 3. Silk Screen

Theory of Operation

Referring to the schematic, Figure 2, the MPX5100 pressure sensor is connected to PORT D bit 5 of the microprocessor. This port is an input to the on-chip 8 bit analog to digital converter. The pressure sensor provides a signal output to the microprocessor of approximately 0.5 Vdc at 0 psi to 4.5 Vdc at 15 psi of applied pressure as shown in Figure 4. The input range of the A to D converter is set at approximately 0.3 Vdc to 4.85 Vdc. This compresses the range of the A to D converter around the output range of the sensor to maximize the A to D converter resolution; 0 to 255 counts is the range of the A to D converter. V_{RH} and V_{RL} are the reference voltage inputs to the A to D converter. The resolution is defined by the following:

Analog-to-digital converter count =

$$[(V_{xocr} - V_{RL}) / (V_{RH} - V_{RL})] \cdot 255$$

The count at 0 psi = $[(.5 - .302) / (4.85 - .302)] \cdot 255 \approx 11$

The count at 15 psi = $[(4.5 - .302) / (4.85 - .302)] \cdot 255 \approx 235$

Therefore the resolution = count @ 15 psi – count @ 0 psi or the resolution is $(235 - 11) = 224$ counts. This translates to a system that will resolve to 0.1 psi.

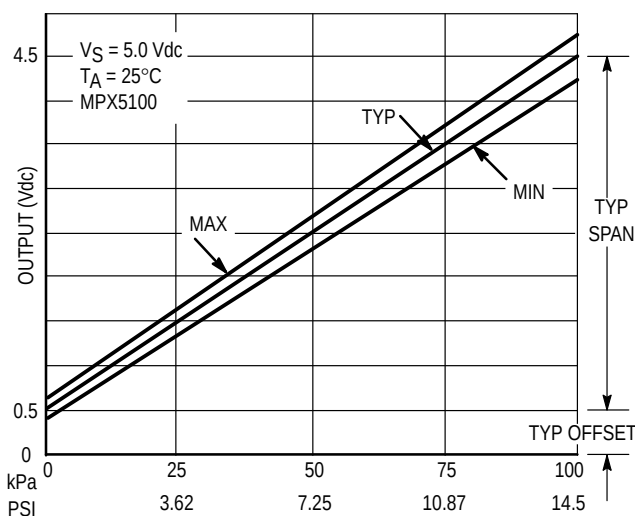


Figure 4. MPX5100 Output versus Pressure Input

The voltage divider consisting of R5 through R7 is connected to the +5 volts powering the system. The output of the pressure sensor is ratiometric to the voltage applied to it. The pressure sensor and the voltage divider are connected to a common supply; this yields a system that is ratiometric. By nature of this ratiometric system, variations in the voltage of the power supplied to the system will have no effect on the system accuracy.

The liquid crystal display is directly driven from I/O ports A, B, and C on the microprocessor. The operation of a liquid crystal display requires that the data and backplane pins must be driven by an alternating signal. This function is provided by a software routine that toggles the data and backplane at approximately a 30 Hz rate.

The microprocessor section of the system requires certain support hardware to allow it to function. The MC34064P-5 (U2) provides an under voltage sense function which is used to reset the microprocessor at system power-up. The 4 MHz crystal (Y1) provides the external portion of the oscillator function for clocking the microprocessor and provides a stable base for time based functions. Jumpers J1 and J2 are examined by the software and are used to "rubber" the slope constant.

OPERATION

The system must be connected to a 5 Vdc regulated power supply. Note the polarity marked on the power terminal J3. Jumpers J1 and J2 must either both be installed or both be removed for the normal slope constant to be used. The pressure port on the MPX5100 sensor must be left open to atmosphere anytime the board is powered-up. As previously stated, the sensor's voltage offset with zero pressure applied is computed at power-up.

You will need to apply power to the system. The LCD will display CAL for approximately 5 seconds. After that time, the LCD will then start displaying pressure.

To improve upon the accuracy of the system, you can change the constant used by the program that constitutes the span of the sensor. You will need an accurate test gauge to measure the pressure applied to the sensor. Anytime after the display has completed the zero calculation (after CAL is no longer displayed), apply 15.0 PSI to the sensor. Make sure that jumpers J1 and J2 are either both installed or both removed. Referring to Table 2, you can increase the displayed value by installing J1 and removing J2. Conversely, you can decrease the displayed value by installing J2 and removing J1.

J1	J2	Action
IN	IN	USE NORMAL SPAN CONSTANT
OUT	OUT	USE NORMAL SPAN CONSTANT
OUT	IN	DECREASE SPAN CONSTANT APPROXIMATELY 1.5%
IN	OUT	INCREASE SPAN CONSTANT APPROXIMATELY 1.5%

Table 2.

SOFTWARE

The source code, compiler listing, and S-record output for the software used in this system are available on the Motorola Freeware Bulletin Board Service in the MCU directory under the filename DEVB-114.ARC. To access the bulletin board you must have a telephone line, a 300, 1200 or 2400 baud modem and a terminal or personal computer. The modem must be compatible with the Bell 212A standard. Call 1-512-891-3733 to access the Bulletin Board Service.

The software for the system consists of several modules. Their functions provide the capability for system calibration as well as displaying the pressure input to the MPX5100 transducer.

Figure 5 is a flowchart for the program that controls the system.

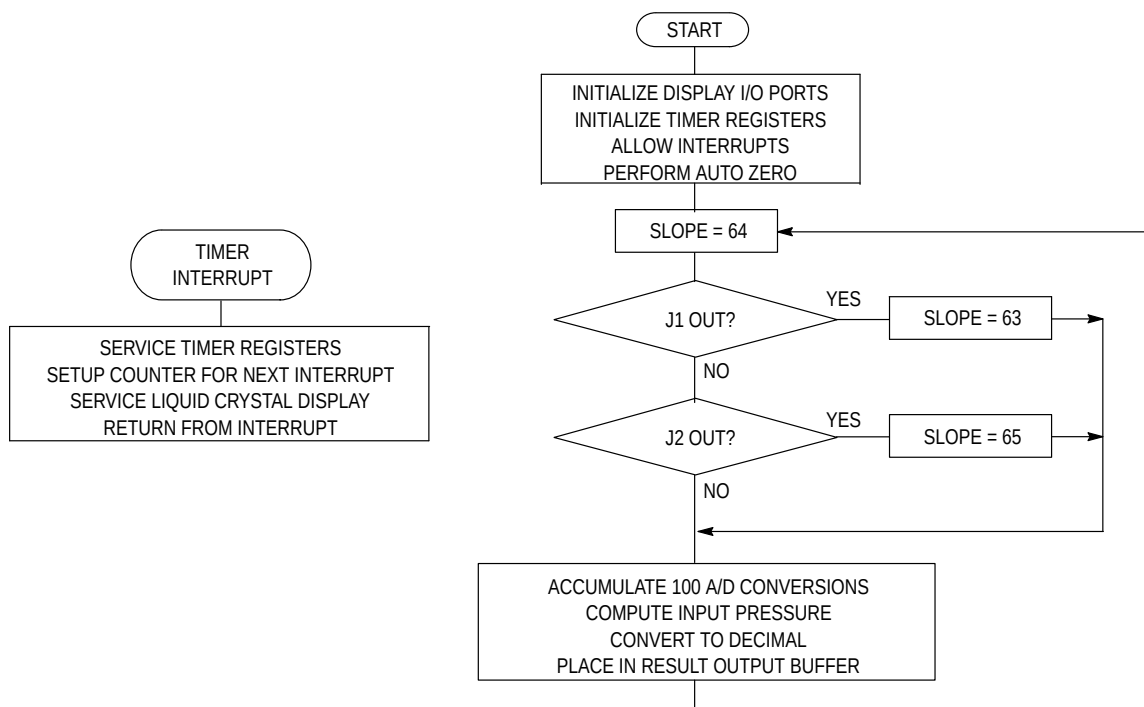


Figure 5. DEVB-114 Software Flowchart

The 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 approximately a 20 ms loop.

read_a2d() Performs one hundred reads on the analog to digital converter on multiplexer channel 5 and returns the accumulation.

fixcompare() Services the internal timer for 30 ms timer compare interrupts.

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

initio() Sets up the microcomputer's I/O ports, timer, allows processor interrupts, and calls `adzero()`.

adzero() This routine is necessary at power-up time because it delays the power supply and allows the

transducer to 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. It is then output to the display.

display_psi() This routine is called from '`main()`'. The analog to digital converter routine is called, the pressure is calculated, and the pressure applied to the sensor is displayed. The loop then repeats.

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

SOFTWARE SOURCE/ASSEMBLY PROGRAM CODE

```
#pragma option v ;
/*
```

```
rev 1.1 code rewritten to use the MC68HC705B5 instead of the
MC68HC805B6. WLL 6/17/91
```

```
THE FOLLOWING 'C' SOURCE CODE IS WRITTEN FOR THE DEVB-114 DEMONSTRATION
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 8/5/90
MOTOROLA, SPS */
```

```
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 acnthi @ 0x1A; /* Alternate Count Reg (Hi-$1A, Lo-$1B) */
001B      #pragma portrw acntlo @ 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)*/

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

/*****
1EFE 74      #pragma mor @ 0x1EFE = 0x74; /* this disables the watchdog counter and does not
add pull-down resistors on ports B and C */

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 functio */

0000      registera ac; /* 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 0060 unsigned long i,j; /* counter for loops */

0062      int k; /* misc variable */

struct bothbytes
{ int hi;
int lo;
};

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

0063 0002 union isboth q; /* used for timer set-up */

```

```

/*****
                                /* code starts here */
*****/
/* these interrupts are not used...give them a graceful return if for
   some reason one occurs */

1FFC 08 12      _SWI(){}
0812 80          RTI
1FFA 08 13      IRQ(){}
0813 80          RTI
1FF8 08 14      TIMERCAP(){}
0814 80          RTI
1FF4 08 15      TIMEROV(){}
0815 80          RTI
1FF2 08 16      SCI(){}
0816 80          RTI

/*****

void delay(void) /* just hang around for a while */
{
0817 4F          CLRA          for (i=0; i<20000; ++i);
0818 3F 57          CLR          $57
081A B7 58          STA          $58
081C B6 57          LDA          $57
081E B7 5E          STA          $5E
0820 B6 58          LDA          $58
0822 B7 5F          STA          $5F
0824 B6 5F          LDA          $5F
0826 A0 20          SUB          #$20
0828 B6 5E          LDA          $5E
082A A2 4E          SBC          #$4E
082C 24 08          BCC          $0836
082E 3C 5F          INC          $5F
0830 26 02          BNE          $0834
0832 3C 5E          INC          $5E
0834 20 EE          BRA          $0824
0836 81          RTS          }
*****/

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

0837 3F 56          CLR          $56      atodtemp=0; /* zero for accumulation */
0839 3F 55          CLR          $55
083B 4F          CLRA          for ( adcnt = 0 ; adcnt<100; ++adcnt) /* do 100 a/d conversions */
083C B7 5B          STA          $5B
083E B6 5B          LDA          $5B
0840 A8 80          EOR          #$80
0842 A1 E4          CMP          #$E4
0844 24 21          BCC          $0867

```



```

0846 A6 25    LDA    #$25
0848 B7 09    STA    $09
084A 0F 09 FD BRCLR  7,$09,$084A
084D B6 08    LDA    $08
084F 3F 57    CLR    $57
0851 B7 58    STA    $58
0853 BB 56    ADD    $56
0855 B7 58    STA    $58
0857 B6 57    LDA    $57
0859 B9 55    ADC    $55
085B B7 57    STA    $57
085D B7 55    STA    $55
085F B6 58    LDA    $58
0861 B7 56    STA    $56

0863 3C 5B    INC    $5B
0865 20 D7    BRA    $083E
0867 B6 56    LDA    $56
0869 B7 58    STA    $58
086B B6 55    LDA    $55
086D B7 57    STA    $57
086F 3F 66    CLR    $66
0871 A6 64    LDA    #$64
0873 B7 67    STA    $67
0875 CD 0A 5E JSR    $0A5E
0878 CD 0A 8F JSR    $0A8F
087B BF 55    STX    $55
087D B7 56    STA    $56
087F 81      RTS

                                {
                                adstat = 0x25; /* convert on channel 5 */
                                while (!(adstat & 0x80)); /* wait for a/d to complete */
                                atodtemp = addata + atodtemp;

                                }

                                atodtemp = atodtemp/100;

                                return atodtemp;
                                }

/*****/

void fixcompare (void) /* sets-up the timer compare for the next interrup */
{
    q.b.hi = tcnthi;
    q.b.lo = tcntlo;
    q.l += 7500; /* ((4mhz xtal/2)/4) = counter period = 2us.*7500 = 15ms.*/

    ocmphi1 = q.b.hi;
    ac=tsr;
    ocmplo1 = q.b.lo;
}

/*****/

void TIMERCMP (void) /* timer service module */
{
1FF6 08 9B

```

AN1305

```

089B 33 02    COM    $02                portc =~ portc;          /* service the lcd */
089D 33 01    COM    $01                portb =~ portb;
089F 33 00    COM    $00                porta =~ porta;
08A1 AD DD    BSR    $0880              fixcompare();
08A3 80        RTI                      }

/*****

void adzero(void) /* called by initio() to save initial xdcr's zero
                  pressure offset voltage output */
{
    for ( j=0; j<20; ++j) /* give the sensor time to "warm-up" and the

08A4 4F        CLRA
08A5 3F 57    CLR    $57
08A7 B7 58    STA    $58
08A9 B6 57    LDA    $57
08AB B7 60    STA    $60
08AD B6 58    LDA    $58
08AF B7 61    STA    $61
08B1 B6 61    LDA    $61
08B3 A0 14    SUB    #$14
08B5 B6 60    LDA    $60
08B7 A2 00    SBC    #$00
08B9 24 0B    BCC    $08C6

                                power supply time to settle down */
                                {
08BB CD 08 17 JSR    $0817          delay();
                                }

08BE 3C 61    INC    $61
08C0 26 02    BNE    $08C4
08C2 3C 60    INC    $60
08C4 20 EB    BRA    $08B1
08C6 CD 08 37 JSR    $0837          xdcr_offset = read_a2d();
08C9 3F 5C    CLR    $5C
08CB B7 5D    STA    $5D
08CD 81        RTS

}

*****/

void initio (void) /* setup the I/O */
{
    adstat = 0x20; /* power-up the A/D */

08CE A6 20    LDA    #$20
08D0 B7 09    STA    $09
08D2 3F 02    CLR    $02
08D4 3F 01    CLR    $01
08D6 3F 00    CLR    $00
08D8 A6 FF    LDA    $FF
08DA B7 06    STA    $06
08DC B7 05    STA    $05
08DE B7 04    STA    $04
08E0 B6 13    LDA    $13
08E2 3F 1E    CLR    $1E
08E4 3F 16    CLR    $16
08E6 B6 1F    LDA    $1F
08E8 AD 96    BSR    $0880

    ddra = ddrb = ddrc = 0xff;

    ac=tsr; /* dummy read */
    ocmphi1 = ocmphi2 = 0;

    ac = ocmphi2; /* clear out output compare 2 if it happens to be set */
    fixcompare(); /* set-up for the first timer interrupt */

```

```

08EA A6 40    LDA    #$40          tcr = 0x40;
08EC B7 12    STA    $12
08EE 9A       CLI
                                CLI; /* let the interrupts begin ! */
                                /* write CAL to the display */
                                portc = 0xcc; /* C */

08EF A6 CC    LDA    $SCC
08F1 B7 02    STA    $02
08F3 A6 BE    LDA    $BE          portb = 0xbe; /* A */
08F5 B7 01    STA    $01
08F7 A6 C4    LDA    $C4          porta = 0xc4; /* L */
08F9 B7 00    STA    $00
08FB AD A7    BSR    $08A4
08FD 81       RTS                adzero();
                                }

/*****
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 suppresses the value and writes it to
the display ports.
The argument value range is 0..65535 decimal. */

0069          {
08FE BF 69    STX     $69
0900 B7 6A    STA     $6A
006B
006C
0902 4F       CLRA
0903 B7 6B    STA     $6B
0905 B6 6B    LDA     $6B
0907 A1 05    CMP     #$05
0909 24 07    BCC     $0912

                                {
                                digit[i] = 0x0; /* put blanks in all digit positions */
                                }

090B 97       TAX
090C 6F 50    CLR     $50,X

                                for ( i=0; i < 4; ++i )

090E 3C 6B    INC     $6B
0910 20 F3    BRA     $0905

                                for ( i=0; i < 4; ++i )

0912 4F       CLRA
0913 B7 6B    STA     $6B
0915 B6 6B    LDA     $6B
0917 A1 04    CMP     #$04
0919 24 70    BCC     $098B

                                {
                                if ( arg >= dectable [i] )

091B 97       TAX
091C 58       LSLX
091D D6 08 0B LDA     $080B,X
0920 B1 6A    CMP     $6A
0922 26 07    BNE     $092B
0924 D6 08 0A LDA     $080A,X
0927 B1 69    CMP     $69
0929 27 5C    BEQ     $0987

                                {
                                l = dectable[i];

092B BE 6B    LDX     $6B
092D 58       LSLX
092E D6 08 0A LDA     $080A,X

```

AN1305

```

0931 B7 6C      STA      $6C
0933 D6 08 0B   LDA      $080B,X
0936 B7 6D      STA      $6D
0938 B6 6A      LDA      $6A          digit[i] = arg / 1;
093A B7 58      STA      $58
093C B6 69      LDA      $69
093E B7 57      STA      $57
0940 B6 6C      LDA      $6C
0942 B7 66      STA      $66
0944 B6 6D      LDA      $6D
0946 B7 67      STA      $67
0948 CD 0A 5E   JSR      $0A5E
094B CD 0A 8F   JSR      $0A8F
094E BF 57      STX      $57
0950 B7 58      STA      $58
0952 BE 6B      LDX      $6B
0954 E7 50      STA      $50,X
0956 BE 6B      LDX      $6B          arg = arg-(digit[i] * 1);
0958 E6 50      LDA      $50,X
095A 3F 57      CLR      $57
095C B7 58      STA      $58
095E B6 6C      LDA      $6C
0960 B7 66      STA      $66
0962 B6 6D      LDA      $6D
0964 B7 67      STA      $67
0966 CD 0A 3F   JSR      $0A3F
0969 BF 57      STX      $57
096B B7 58      STA      $58
096D 33 57      COM      $57
096F 30 58      NEG      $58
0971 26 02      BNE      $0975
0973 3C 57      INC      $57
0975 B6 58      LDA      $58
0977 BB 6A      ADD      $6A
0979 B7 58      STA      $58
097B B6 57      LDA      $57
097D B9 69      ADC      $69
097F B7 57      STA      $57
0981 B7 69      STA      $69
0983 B6 58      LDA      $58
0985 B7 6A      STA      $6A
                                }
                                }

0987 3C 6B      INC      $6B
0989 20 8A      BRA      $0915
098B B6 6A      LDA      $6A          digit[i] = arg;
098D B7 58      STA      $58
098F B6 69      LDA      $69
0991 B7 57      STA      $57
0993 BE 6B      LDX      $6B
0995 B6 58      LDA      $58
0997 E7 50      STA      $50,X

                                /* now zero suppress and send the lcd pattern to the display */
0999 9B          SEI      SEI;

```

```

099A 3D 50    TST    $50
099C 26 04    BNE    $09A2
099E 3F 02    CLR    $02
09A0 20 07    BRA    $09A9
09A2 BE 50    LDX    $50
09A4 D6 08 00 LDA    $0800,X
09A7 B7 02    STA    $02
09A9 3D 50    TST    $50
09AB 26 08    BNE    $09B5
09AD 3D 51    TST    $51
09AF 26 04    BNE    $09B5
09B1 3F 01    CLR    $01
09B3 20 07    BRA    $09BC
09B5 BE 51    LDX    $51
09B7 D6 08 00 LDA    $0800,X
09BA B7 01    STA    $01
09BC BE 52    LDX    $52
09BE D6 08 00 LDA    $0800,X
09C1 4C      INCA
09C2 B7 00    STA    $00
09C4 9A      CLI
09C5 CD 08 17 JSR    $0817
09C8 81      RTS

if ( digit[0] == 0 ) /* leading zero suppression */
    portc = 0;
    else
        portc = ( lcdtab[digit[0]] ); /* 100's digit */

        if ( digit[0] == 0 && digit[1] == 0 )

            portb=0;
            else
                portb = ( lcdtab[digit[1]] ); /* 10's digit */

                porta = ( lcdtab[digit[2]]+1 ); /* 1's digit + decimal point */

CLI;
        delay();
}

/*****/

void display_psi(void)
/* At power-up it is assumed that the pressure port of the sensor
is open to atmosphere. The code in initio() delays for the
sensor and power to stabilize. One hundred A/D conversions are
averaged and divided by 100. The 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. That result is multiplied by a constant to yield
pressure in PSI * 10 to yield a "decimal point".
*/
{
    while(1)
    {
09C9 3F 59    CLR    $59
09CB A6 40    LDA    #$40
09CD B7 5A    STA    $5A
09CF B6 03    LDA    $03
09D1 A4 C0    AND    #$C0
09D3 B7 62    STA    $62

        slope = 64;

        k = portd & 0xc0; /* this lets us "rubber" the slope to closer fit

                                the slope of the sensor */
        if ( k == 0x80 ) /* J2 removed, J1 installed */

            slope = 65;

            if ( k == 0x40 ) /* J1 removed, J2 installed */

```

AN1305

09E1	A1 40	CMP	#\$40	
09E3	26 06	BNE	\$09EB	
09E5	3F 59	CLR	\$59	slope = 63;
09E7	A6 3F	LDA	#\$3F	
09E9	B7 5A	STA	\$5A	
09EB	CD 08 37	JSR	\$0837	/* else both jumpers are removed or installed... don't change the slope */
09EE	3F 55	CLR	\$55	atodtemp = read_a2d(); /* atodtemp = raw a/d (0..255) */
09F0	B7 56	STA	\$56	
09F2	B0 5D	SUB	\$5D	if (atodtemp <= xdcr_offset)
09F4	B7 58	STA	\$58	
09F6	B6 5C	LDA	\$5C	
09F8	A8 80	EOR	#\$80	
09FA	B7 57	STA	\$57	
09FC	B6 55	LDA	\$55	
09FE	A8 80	EOR	#\$80	
0A00	B2 57	SBC	\$57	
0A02	BA 58	ORA	\$58	
0A04	22 08	BHI	\$0A0E	
0A06	B6 5C	LDA	\$5C	atodtemp = xdcr_offset;
0A08	B7 55	STA	\$55	
0A0A	B6 5D	LDA	\$5D	
0A0C	B7 56	STA	\$56	
0A0E	B6 56	LDA	\$56	atodtemp -= xdcr_offset; /* remove the offset */
0A10	B0 5D	SUB	\$5D	
0A12	B7 56	STA	\$56	
0A14	B6 55	LDA	\$55	
0A16	B2 5C	SBC	\$5C	
0A18	B7 55	STA	\$55	
0A1A	B6 56	LDA	\$56	atodtemp *= slope; /* convert to psi */
0A1C	B7 58	STA	\$58	
0A1E	B6 55	LDA	\$55	
0A20	B7 57	STA	\$57	
0A22	B6 59	LDA	\$59	
0A24	B7 66	STA	\$66	
0A26	B6 5A	LDA	\$5A	
0A28	B7 67	STA	\$67	
0A2A	CD 0A 3F	JSR	\$0A3F	
0A2D	BF 55	STX	\$55	
0A2F	B7 56	STA	\$56	
0A31	CD 08 FE	JSR	\$08FE	cvt_bin_dec(atodtemp); /* convert to decimal and display */
0A34	20 93	BRA	\$09C9	}
0A36	81	RTS		}
/*****				
main()				
{				
0A37	CD 08 CE	JSR	\$08CE	initio(); /* set-up the processor's i/o */
0A3A	AD 8D	BSR	\$09C9	display_psi();
0A3C	20 FE	BRA	\$0A3C	while(1); /* should never get here */
0A3E	81	RTS		}
0A3F	BE 58	LDX	\$58	
0A41	B6 67	LDA	\$67	

0A43	42	MUL	
0A44	B7 70	STA	\$70
0A46	BF 71	STX	\$71
0A48	BE 57	LDX	\$57
0A4A	B6 67	LDA	\$67
0A4C	42	MUL	
0A4D	BB 71	ADD	\$71
0A4F	B7 71	STA	\$71
0A51	BE 58	LDX	\$58
0A53	B6 66	LDA	\$66
0A55	42	MUL	
0A56	BB 71	ADD	\$71
0A58	B7 71	STA	\$71
0A5A	97	TAX	
0A5B	B6 70	LDA	\$70
0A5D	81	RTS	
0A5E	3F 70	CLR	\$70
0A60	5F	CLR	
0A61	3F 6E	CLR	\$6E
0A63	3F 6F	CLR	\$6F
0A65	5C	INC	
0A66	38 58	LSL	\$58
0A68	39 57	ROL	\$57
0A6A	39 6E	ROL	\$6E
0A6C	39 6F	ROL	\$6F
0A6E	B6 6E	LDA	\$6E
0A70	B0 67	SUB	\$67
0A72	B7 6E	STA	\$6E
0A74	B6 6F	LDA	\$6F
0A76	B2 66	SBC	\$66
0A78	B7 6F	STA	\$6F
0A7A	24 0D	BCC	\$0A89
0A7C	B6 67	LDA	\$67
0A7E	BB 6E	ADD	\$6E
0A80	B7 6E	STA	\$6E
0A82	B6 66	LDA	\$66
0A84	B9 6F	ADC	\$6F
0A86	B7 6F	STA	\$6F
0A88	99	SEC	
0A89	59	ROL	
0A8A	39 70	ROL	\$70
0A8C	24 D8	BCC	\$0A66
0A8E	81	RTS	
0A8F	53	COM	
0A90	9F	TXA	
0A91	BE 70	LDX	\$70
0A93	53	COM	
0A94	81	RTS	
1FFE	0A 37		

AN1305

SYMBOL TABLE

LABEL	VALUE	LABEL	VALUE	LABEL	VALUE	LABEL	VALUE
IRQ	0813	SCI	0816	TIMERCAP	0814	TIMERCMP	089B
TIMEROV	0815	__LDIV	0A5E	__LongIX	0066	__MUL	0000
__MUL16x16	0A3F	__RDIV	0A8F	__RESET	1FFE	__STARTUP	0000
__STOP	0000	__SWI	0812	__WAIT	0000	__longAC	0057
acnthi	001A	acntlo	001B	adcnt	005B	addata	0008
adstat	0009	adzero	08A4	arg	0069	atodtemp	0055
b	0000	bothbytes	0002	cvt_bin_dec	08FE	ddra	0004
ddrb	0005	ddrc	0006	dectable	080A	delay	0817
digit	0050	display_psi	09C9	eeclk	0007	fixcompare	0880
hi	0000	i	005E	icaphi1	0014	icaphi2	001C
icaplo1	0015	icaplo2	001D	initio	08CE	isboth	0002
j	0060	k	0062	l	0000	lcdtab	0800
lo	0001	main	0A37	misc	000C	ocmphi1	0016
ocmphi2	001E	ocmplo1	0017	ocmplo2	001F	plma	000A
plmb	000B	porta	0000	portb	0001	portc	0002
portd	0003	q	0063	read_a2d	0837	scibaud	000D
scicnt11	000E	scicnt12	000F	scidata	0011	scistat	0010
slope	0059	tcnthi	0018	tcntlo	0019	tcr	0012
tsr	0013	xdcr_offset	005C				

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

```

0100 : -----
0140 : -----
0180 : -----
01C0 : -----X-

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 XXXXX-----
0AC0 : -----

1F00 : -----
1F40 : -----
1F80 : -----
1FC0 : -----XXXXXXXXXX

```

All other memory blocks unused.

Errors : 0
Warnings : 0

Motorola reserves the right to make changes without further notice to any products herein. Motorola makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Motorola assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters can and do vary in different applications. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Motorola does not convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part. Motorola and  are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

Literature Distribution Centers:

USA: Motorola Literature Distribution; P.O. Box 20912; Phoenix, Arizona 85036.

EUROPE: Motorola Ltd.; European Literature Centre; 88 Tanners Drive, Blakelands, Milton Keynes, MK14 5BP, England.

JAPAN: Nippon Motorola Ltd.; 4-32-1, Nishi-Gotanda, Shinagawa-ku, Tokyo 141, Japan.

ASIA PACIFIC: Motorola Semiconductors H.K. Ltd.; Silicon Harbour Center, No. 2 Dai King Street, Tai Po Industrial Estate, Tai Po, N.T., Hong Kong.



AN1305/D

