Order by AN1921/D
(Motorola Order Number)
Rev. 1.0, 04/2001

General DSP568xx Interface Examples using the Embedded SDK

Joseph R. Pasek

Contents

1. Introduction.....................................1
2. LCD and Keypad User Interface Description.................................1
3. Interfacing to Serial Devices Using the SPI Port................................7
4. Integrating Devices With the EVM Board.......................................15
Appendices
A. Details of NewMedia's Serial LCD+.......................................20
B. TI's TLC2543...............................24
C. Header Files TLC2543.h and lcd.h 27
D. Determining the Pressure Transducer's Altitude Pressure Adjustment...............................28
E. Motorola's MPX4115A Integrated Pressure Sensor........................29

1. Introduction

The purpose of this application note is to describe the process of interfacing different devices to the Motorola DSP56824 processor using Motorola's Embedded SDK and Metrowerks' C compiler. This is in addition to the devices available on the DSP's EVM card described in the associated SDK documentation. Examples include interfacing to LCDs, Keypads, ADCs and pressure sensors.

It is assumed that the reader has some familiarity with both the Metrowerks' IDE and Motorola's Embedded Software Development Kit (SDK).

2. LCD and Keypad User Interface Description

As the DSP568xx family assumes more microcontroller roles, it must be capable of working in the embedded environment, on occasion requiring some form of user interface. This user interface is usually characterized by some combination of LCD, LEDs, buttons, and keypad. The combination used in this note is the LCD and 4x4 keypad.

The LCD used here is NetMedia's Serial LCD+, which is a 4x20 LCD serial display. It is an off-the-shelf unit (see **Figure 1**) with several interesting features. In addition to the LCD display, the Serial LCD+ provides a keypad interface, an eight channel 10-bit A/D converter, an EEPROM and RS-232 interface.

© Motorola, Inc., 2001

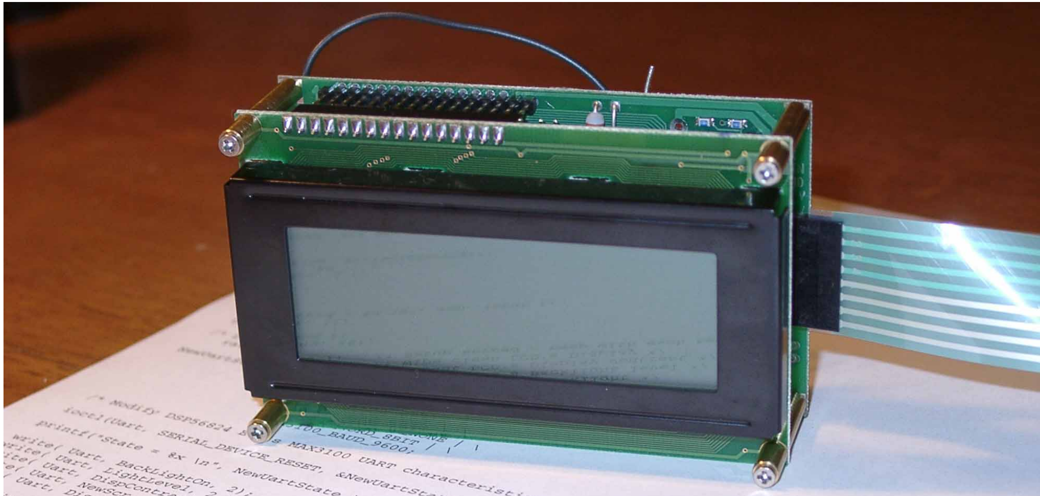


Figure 1. NetMedia's 4x20 LCD Device with RS-232 Serial and Keypad Support

The keypad is a membrane-encased, low-cost touch pad and is shown in **Figure 2**.

Figure 2. Low Cost Membrane Covered 4x4 Keypad that Interfaces to LCD+

All examples described here use the Motorola DSP56824 EVM board, illustrated in **Figure 3**. The board provides a developer with the means of both learning to know the processor featured and the capability to prototype designs. The board provides both PC parallel port and JTAG interfaces for software development.

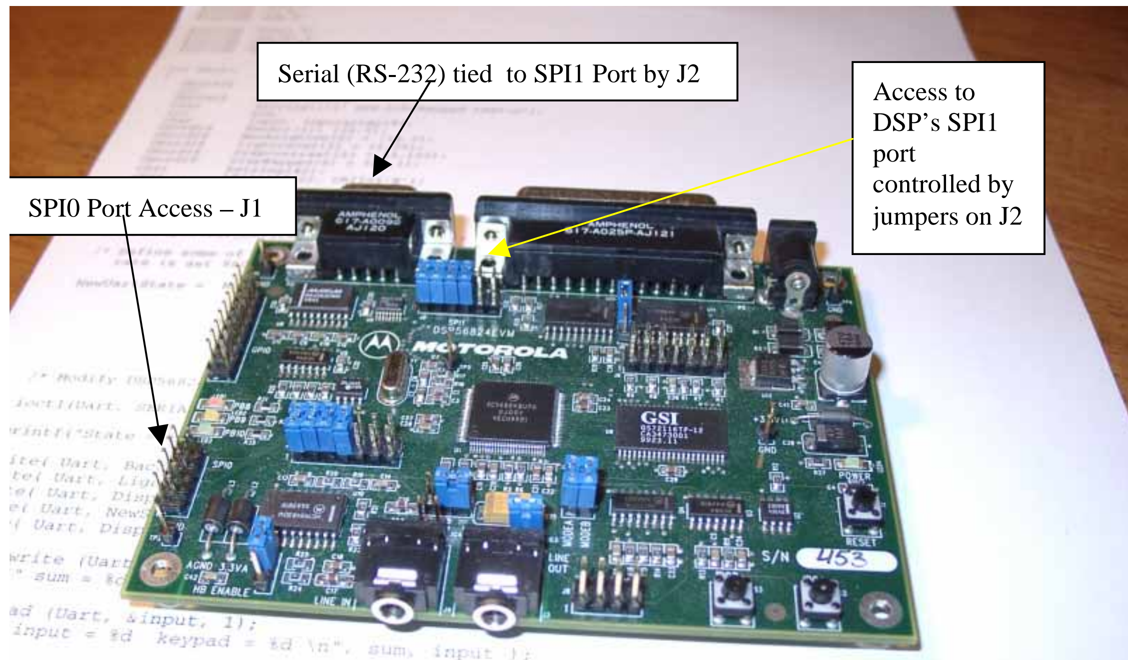


Figure 3. The DSP56824EVM board used in this note, with annotations added

The DSP56824 provides a number of interfaces: Port A provides for access to external memory located on the EVM. It also provides another interface, referred to as Port B, which has 16 general purpose I/O lines. Each of the lines are user-controlled and can be directed as either input or output. The lower eight lines can generate interrupts based on the rising or falling edge of a signal. A third interface, referred to as Port C, can be either GPIO lines or can be allocated to timers, two SPI (Serial Peripheral Interfaces) ports and an SSI (Synchronous Serial Interface) port.

The SPI port is an independent, serial communication subsystem that allows the DSP56824 to communicate synchronously with peripheral devices such as LCD drivers, A/D and D/A subsystems, and microprocessors. The SPI can be configured as either a master or a slave device with high data rates. In master mode, a transfer is initiated when data is written to the SPI data register (SPDR). In slave mode, a transfer is initiated by the reception of a clock signal.

Clock control logic allows a selection of clock polarity and a choice of two fundamentally different clocking protocols to accommodate most available synchronous serial peripheral devices. In some cases, the phase and polarity are changed between transfers to allow a master device to communicate with peripheral slaves having different requirements. When the SPI is configured as a master, software selects one of eight different bit rates for the clock.

On the DSP56824EVM board, the chip's SPI1 port is optionally interfaced by means of the jumpers placed at the board's J2 header to MAXIM's MAX3100 UART. In this case, the MAX3100 provides an interface between the SPI and RS-232 interfaces.

The EVM provides a 9-pin Sub-D connector to attach a serial cable. Direct access to the SPI1 is possible by removing the jumpers at J2. Table 1 describes the pins found at J2.

Table 1: Signal description at J2 header between SPI1 and MAX3100 Uart

J2			
Pin #	**DSP Signal**	**Pin #**	**UART Signal**
1	MISO1/PC4	2	DOUT
3	MOSI1/PC5	4	DIN
5	SCK1/PC6	6	SCLK
7	SS1/PC7	8	CS
9	IRQA	10	IRQB
11	GND	12	GND

The Metrowerks' C compiler/IDE for the Motorola DSP568xx is used to produce the example code in C which demonstrates usage of the LCD/keypad. This code is shown in **Code Example 1** and uses Motorola's Embedded SDK serial and EVM board support libraries. The SDK's serial library accommodates the MAX3100 UART chip that is interfaced to the EVM board's DSP56824's SPI1 port. Additional support for the EVM board is provided by the SDK's bsp library; for more details, see the Embedded SDK documentation. The support to these libraries are provided by the included *serial.h* and *bsp.h* header files.

Code Example 1. *LCDsimple.c,* code used to demo DSP control and use of LCD/keypad

```
// LCDsimple.c program tests the interface to a serial port LCD and keypad

#include "port.h"
#include "io.h"
#include "bsp.h"

#include "fcntl.h"
#include "serial.h"
#include "stdio.h"
#include "string.h"


int main()
{
   UWord16     I;
   int         Uart;
   UWord16     NewUartState;
   char        astring[]={" DSP-LCD/Keypad test\n"};
   int         sum;
   char        input, inputarray[8];
   UWord16     NewScr[2]= {12,0};
   UWord16     BackLightOn[2] = {14,0};
   UWord16     LightLevel[2] = {2,70};
   UWord16     DispContrast[2] = {3,100};
   UWord16     DispKeypad[2] = {24,1};
   char        bstring[40];
   char        LF[]={10,0}, CR[]={'E'};
   UWord16     ii;
```

```
UWord16    BS[]={8,0};

/* Open Serial Device */

Uart = open(BSP_DEVICE_NAME_SERIAL_0, 0);

/* Define some of the attributes of the MAX3100 UART - note BAUD
   rate is set to 9600 */

NewUartState =  MAX3100_FIFO_DISABLE | \
                   MAX3100_INT_ENABLE_DATA | \
                   MAX3100_IR_DISABLE | \
                   MAX3100_STOPBIT_1 | \
                   MAX3100_PARITY_NONE | \
                   MAX3100_WORD_8BIT | \
                   MAX3100_BAUD_9600;

/* Modify DSP56824 EVM's MAX3100 UART characteristics */

ioctl(Uart, SERIAL_DEVICE_RESET, &NewUartState);
```

Code Example 1 continues below.

As shown in the code *LCDsimple.c,* processing starts by a call to *open()*, which allocates a handle (UART) to the application for the RS-232 serial port to which the SPI1 is interfaced. The control variable, NewUartState, is initialized to define the attributes of the interface to the MAX3100 UART chip and is directed to the MAX3100 by a call using the function *ioctl()*.

Code Example 1, continued:

```
printf("State = %x \n", NewUartState );

   write( Uart, BackLightOn, 2);   /* Turn-on LCD's Backlight */
   write( Uart, LightLevel, 2 );   /* Adjust LCD's Backlight level */
   write( Uart, DispContrast, 2);  /* Adjust LCD's Display contrast */
   write( Uart, NewScr, 1 );       /* Wipe clean LCD's Display */
   write( Uart, DispKeypad, 2 );   /* Setup keypad - beep with each key
press */

   sum = write (Uart, astring, 16);
   printf(" sum = %d \n", sum );

   sum = read (Uart, &input, 1);
   printf(" input = %d  keypad = %d \n", sum, input );

   sum = 0;

   for ( I=0; I < 10; I++ )
   {
        write( Uart, NewScr, 1 );
        sprintf(astring, " %d ", I );
        write (Uart, astring, strlen(astring));

   }

   strcpy( bstring, "Keypad: hit any key.");
   strcat( bstring, LF );
   write( Uart, bstring, strlen(bstring));
   sum = read (Uart, &input, 1);
```

```
        strcpy( bstring, "Input at keypad ('0000 (CR)' to exit) ");
        strcat( bstring, LF );
        write( Uart, bstring , strlen(bstring));

        strcpy( bstring, "Hit key to start");
        strcat( bstring, LF );
        write( Uart, bstring , strlen(bstring));
        write( Uart, NewScr, 1 );

        /* The following loop polls keypad input */

        ii = 0;
        while ( true )
        {
           read( Uart, &inputarray[ii], 1);
           if ( inputarray[ii] == CR[0]  )
           {
              sprintf(bstring,"\n");
              write ( Uart, bstring, strlen(bstring));
              inputarray[ii] = 0;
              sprintf(bstring, "Input - %s ", inputarray );
              strcat( bstring, LF );
              printf(" %s", bstring );
              ii = 0;
           }
               else
           {
              write( Uart, &inputarray[ii], 1);
             ii++;
           }
           if ( strcmp(inputarray,"0000") == 0 ) break;
        }

        write( Uart, NewScr, 1 );
        strcpy( bstring, " Done! ");
        write(Uart, bstring, strlen(bstring) );
    }
```

Next, a series of calls using *write()* directs commands to the LCD. The commands issued are to:

- Turn on the LCD's backlight
- Adjust the light level
- Adjust the display contrast
- Clear the display area
- Set up the keypad

In this example, selected attributes cause the keypad to both beep with each keypad entry and immediately display (Echos) the entry on the LCD. More detailed information about the LCD used here is found in **Appendix A.**

A call to *write()* next directs the string that announces the test to the LCD. The number of characters passed (sum) is passed to *printf()* and displayed in the IDE's text window.

A call to *read()* yields whatever key was struck on the keypad. The ASCII code of the keypad character struck is passed to *printf()*.

The keypad's key assignments are shown in **Figure 4**.

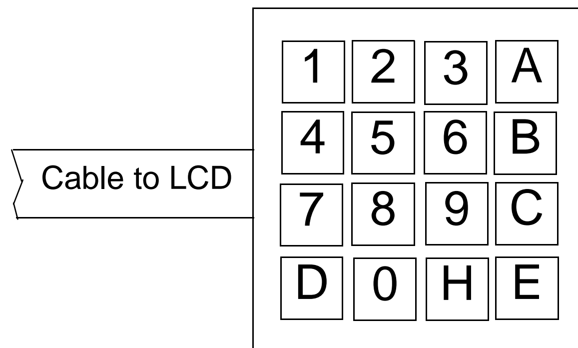


Figure 4. Keypad's Key Assignment Values

A polling loop is executed which takes input from the keypad and displays the value passed to the program in the *printf()* text window. When the user enters a string and presses the E [Enter] key, the string is displayed in the IDE's text window. After entering values, the user exits the loop by entering "0000", followed by [Enter]. The display will clear and the string "Done!" will appear.

3. Interfacing to Serial Devices Using the SPI Port

The previous section described the LCD/keypad interface and operation from the SPI1 port via the MAX3100 UART. The connection to the UART and, in turn, the EVM's RS-232 serial port was established, with the default jumpers left in place at the EVM's J2 header. The DSP's SPI0 is interfaced to a serial EEPROM via the jumpers left in place on the J1 header. Removing the jumpers and connecting wires to pins 1,3,5, and 7 permits the user to directly interface another SPI device to the DSP; see **Table 2**.

Table 2: Signal Description at J1 Header between SPI0 and EEPROM

J1			
Pin #	**DSP Signal**	**Pin #**	**EEPROM Signal**
1	MISO1/PC0	2	SD1
3	MOSI1/PC1	4	SD0
5	SCK1/PC2	6	SCK
7	SS1/PC3	8	CS
9	GND	10	GND

Table 3: Connections between SPI0 (J1) EVM Header and TLC2543 Chip

J1 Header Signal	TLC2543 lines
MISO1/PC0	DATAOUT (16)
MOSI1/PC1	DATA IN (17)
SCK1/PC2	I/O CLOCK (18)
SS1/PC3	CS (15)

The TI TLC2543 SPI device is interfaced to the DSP's SPI0 port by means of the EVM's J1 header pins. The TLC2543 is a MUXed 11-channel, 12-bit analog-to-digital converter. Control of the TLC2543 chip is performed when the application code sends a command word to it. A description of the TLC2543 command word appears in Appendix 2.

The C code is written using Motorola's Embedded SDK's SPI library and the BSP library. Using a prototype board (see **Figure 5**) to support the TLC2543, all required connections are made to provide power, reference voltages and ground to the ADC. See **Figure 6** for a schematic. Jumper wires are placed between J1's 1,3,5, and 7 pins and the CLKIN, DATAIN, DATAOUT, and CS on the TLC2543 (see **Figure 6** and **Table 3**).

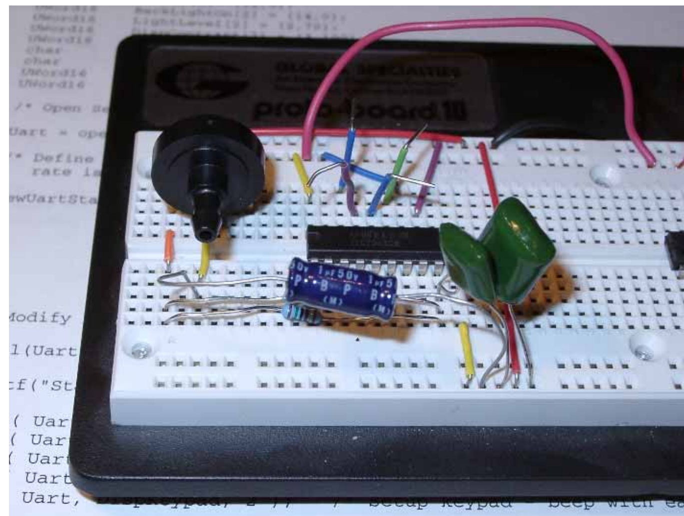


Figure 5. View of Prototype Board Showing the Motorola Absolute Barometric Sensor and TLC5423 ADC

The C code procedure (*InitTLC2453.c*) is used to establish and test the interface between the SPI0 and the ADC. **Code Example 2**, *InitTLC2543.c,* shows operation of TLC2543 from SPI0.

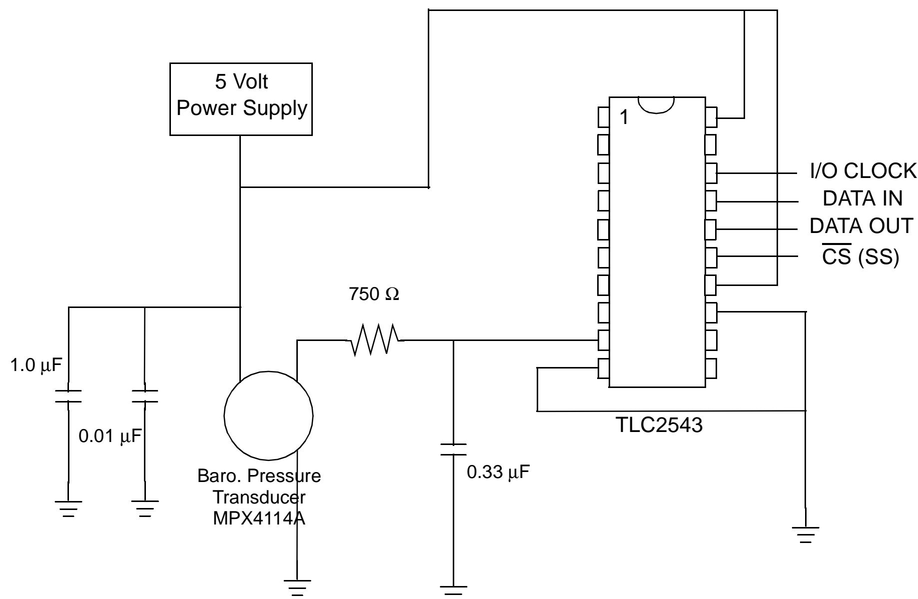


Figure 6. Integrated Pressure and TLC2543 Connections on Prototype Board

Code Example 2. InitTLC2543.c

```
/*  InitTLC2543.c - Procedure used to test interface to TLC2543 -
    12-bit analog-to-digital converter with serial control and 11 analog
    inputs. The DSP56824's SPI0 port is employed. The software is
    developed for the DSP56824 EVM environment. Jan 26, 2001*/

#include "io.h"
#include "fcntl.h"
#include "bsp.h"
#include "spi.h"
#include "stdio.h"
#include "string.h"
#include "port.h"
#include "timer.h"
#include "types.h"
#include "math.h"


void main(void)
{

    spi_sParams     SpiParams;

    int             SPIMaster;

    struct timespec OneMillisecond = {0,1000000};

    UWord16     ADcmd = 0x8c00;
    static UWord16     Datain, DataStore[70];
    Word16     i, ii;
    static Word32    sum, mean;
    static Word16     Vcount;
```

```
static Word16     BaroPress;

SpiParams.bSetAsMaster = 1;    /* SPI0 is set as master */

SPIMaster = open( BSP_DEVICE_NAME_SPI_0, 0, &SpiParams );

/* Set bit clock rate so sampling rate */
ioctl( SPIMaster, SPI_PHI_DIVIDER_32, NULL );

/* Set Data format for 16 bit */
ioctl( SPIMaster, SPI_DATAFORMAT_RAW, NULL );

/* SS can be left low between successive SPI bytes */
ioctl( SPIMaster, SPI_CLK_PHASE_SS_CLEAR, NULL);

/* TLC2543 is commanded to use analog input 0,
   Output data length = 16 bits,
   Output data format = MSB first, unsigned integer */

for(  ;  ;  )
{
  sum = 0;
   for ( i = 0; i < 64; i++ )
   {
    /* send command word to TLC2543  */
    write(SPIMaster, &ADcmd, sizeof(UWord16));

    /* read input from TLC2543  */
    read(SPIMaster, (UWord16 *)&Datain, sizeof(UWord16));

    Datain = Datain >> 4;

    /* process sleeps for Tenth of sec */
    nanosleep( &OneMillisecond, NULL );

    DataStore[i] = Datain;
    sum += Datain;

}
  mean = (sum >> 6);
  Vcount = mean;

/*  The coded equation is an adaption of an
    equation taken from the Integrated Pressure sensor
    Tech Note. The equation relates the voltage out
    (Vout) with the pressure (P) and (Voltage supplied) Vs.

      Vout = Vs * (0.009*P - 0.095)

        where P is pressure in kPa

    Rewriting the equation for P yields,

      P = (Vout/Vs + 0.095)/0.009

 In this set-up for the TLC2543 Vcc = REF+ = Vs. This
 implies that max. voltage is V ~ ADC no. of bit = 2^12
 = 4096. The ADC delivers a count (Vcount) between 0 and 4096.
 This allows the above expression to be written as
```

```
            P = 0.0271*Vcount + 10.555 (kPa)

         Since the sensors measures absolute pressure the units can
         be converted to more recognizable units of millibars. The
         conversion factors is 1 millibar = 100 Pa applied to the
         last equation yields

            P = 0.271*Vcount + 105.55

         which is source of coded equation. The four shifts
         in the coded form of equation produce an approximation
         of 0.2715 */

        BaroPress = (Vcount >> 2) + (Vcount >> 6) + (Vcount >> 8)
                        + (Vcount >> 9 ) + 105;
        }


        close(SPIMaster);
}
```

When compiled and executed on the target EVM, the code in **Code Example 2** shows that data flows bi-directionally between the application code and the ADC chip, using the DSP's SPI0 port.

The Embedded SDK header files *spi.h* and *bsp.h* appear in procedure *InitTLC2543.c* and provide the data structures and other defines required for this code to interface to the processor in general and to the SPI port in particular. The data structure `SpiParams` is defined from the data type `spi_sParams`. The variable SPIMaster is used to store the SPI ports handle. In this example, the variable ADcmd is both defined and set to a value 0x8c00. ADcmd is the command word sent to the TLC2543 ADC chips data input. In the upper 8-bit portion of the word, it specifies use of the eighth analog input port on the ADC; a 16-bit word format, MSB first; and unsigned integer format for output.

To indicate that the DSP's SPI0 port will be the master device, a field (.bSetAsMaster) in the data structure `SpiParams` is set . A call is made to *open()* to allocate the SPI0 port, a handle to the port is returned in the variable SPIMaster. Several calls to the procedure *ioctl()* are used to further refine the SPI0 port's attributes. The first call sets the bit block rate from the PHI clock by providing a divisor term, SPI_PHI_DIVIDER_32. This controls the clock rate of the timing pulses by dividing the clock rate by 32. Next, the command word SPI_DATAFORMAT_RAW specifies that data transfers between the SPI and the slave device will be 16 bits long. The next call to *ioctl()* specifies that the generated SS signal directed to the chip's $\overline{\text{CS}}$ port be left low between the word's byte components.

All the SPI port's attributes have been set. An infinite loop now follows to exercise the link between the SPI0 and the TLC2543 ADC. To average out the possible errors in the reading of the integrated pressure sensor, MPX4115A, 64 measurements of the ADC are done, one millisecond apart. For a description of the MPX4115A integrated pressure sensor, see **Appendix E.**

The loop that performs the actual measurement consists of a call to *write()* to send the variable ADcmd to the TLC2543. A call to *read()* follows to read the previous sampling period's count from the sensor. The result is placed in Datain. The data (Datain) read must be shifted to the right by 4 bits, since it is a 12-bit result placed in a 16-bit word. A call to *nanosleep()* puts the process to sleep for one millisecond.

Using the IDE's debug capability, the last 64 readings are stored in the array DataStore[] for user review. The current sample is also accumulated in the variable sum to perform a batch averaging when 64 recent samples have been read.

Outside the 64 sample loop, the mean is determined and placed in the variable Vcount. Next, the current atmospheric or barometric pressure (BaroPress) is computed.

The coded equation is an adaptation of an equation taken from the MPX4115A Integrated Pressure Sensor Tech Note. The equation relates the voltage out (Vout) with the pressure (P) in kiloPascals and (Voltage supplied) Vs.

$$Vout = Vs * (0.009*P - 0.095)$$

where P is pressure in kPa

Rewriting the equation for P yields,

$$P = (Vout/Vs + 0.095)/0.009$$

In this set-up for the TLC2543, Vcc = REF+ = Vs. This implies that maximum voltage is V ~ ADC number of bits = 2^12 = 4096. The ADC delivers a count (Vcount) between 0 and 4096. This allows the above expression to be written as:

$$P = 0.0271*Vcount + 10.555 \quad (kPa)$$

Since the sensor measures absolute pressure, the units can be converted to the more recognizable units of millibars. The conversion factor is 1 millibar = 100 kPa. When applied to the last equation, it yields:

$$P = 0.271*Vcount + 105.55$$

which is the source of the expression implemented in the code. The coded expression employs 4 shifts and 4 adds to yield the pressure in millibars.

There are several ways to visualize what is happening. The first is to halt (not kill) the process from the Metrowerks' IDE debug window. Once the process is halted, the contents of the DataStore[] array can be viewed by selecting the View menu and selecting the Global Variables Window.

After the Global Variables Window appears, select *Inttlc2543* from the list on the left. In the right half of the window, select the plus sign to the left of the listed variable name, DataStore, and the current contents of the array will appear. The range of values to be found there will fall between 3000 and 4000, depending on the user's altitude above sea level and weather conditions. If the DataStore[] does not contain a number in this range, check the connection between the EVM's SPI0 header and the ADC with the schematic in **Figure 6** and connection references in **Table 3**. The computed atmospheric pressure (BaroPressure) is also found in the right side of the Global Variables Window. The expected range should be between 800 and 1100.

Another means of checking the proper operation of this test requires access to an oscilloscope and leaving the test procedure executing on the EVM board. Placing a probe on the I/O clock line and triggering on the rising edge of the I/O CLOCK line pulse, should yield a signal similar to that shown on the lower line in **Figure 7**, which represents the SPI-generated clock pulses sent to ADC. Leaving the first probe on the I/O clock line (triggering on the clock pulses) and placing a second probe on the SPI0 MOSI line, will yield the signals seen in **Figure 8**, where the command byte is seen on the upper line and ADcmd =0x8C00.

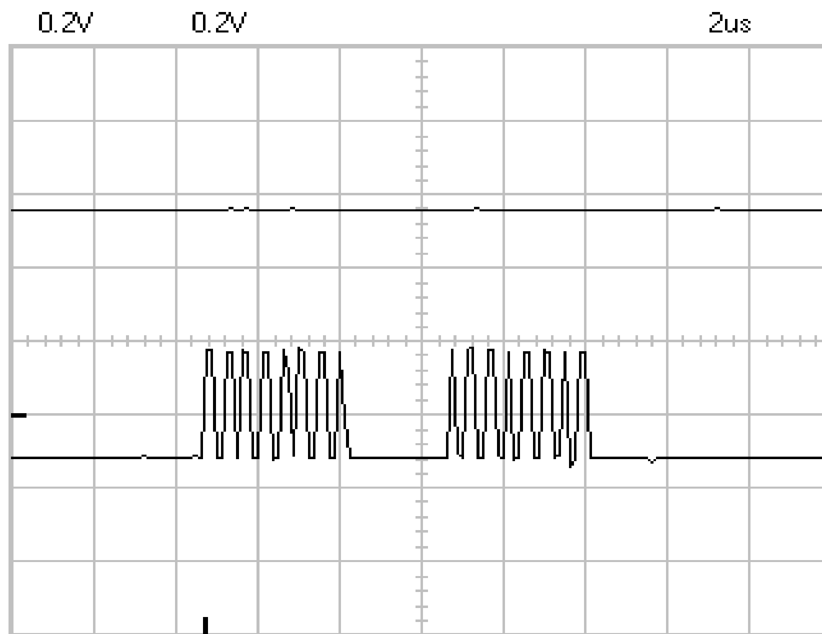


Figure 7. SPI-generated Clock Pulses

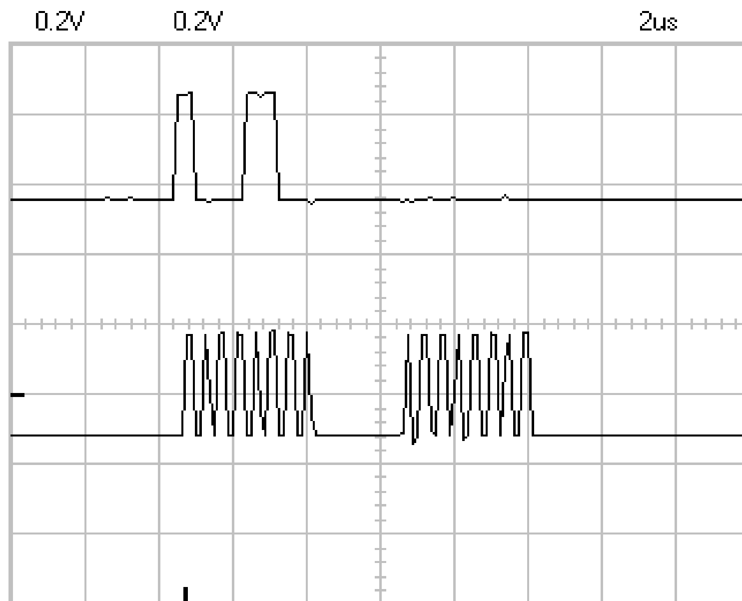


Figure 8. Command Byte (ADcmd = 0x8C00)

Again triggering on the clock lines' pulses and moving the second probe to the SPI0 header's, Slave Select (SS) line yields the signal seen in **Figure 9**. The SS is applied to the TLC2543 $\overline{CS}$ pin. When the signal goes low, processing is enabled on the ADC chip.

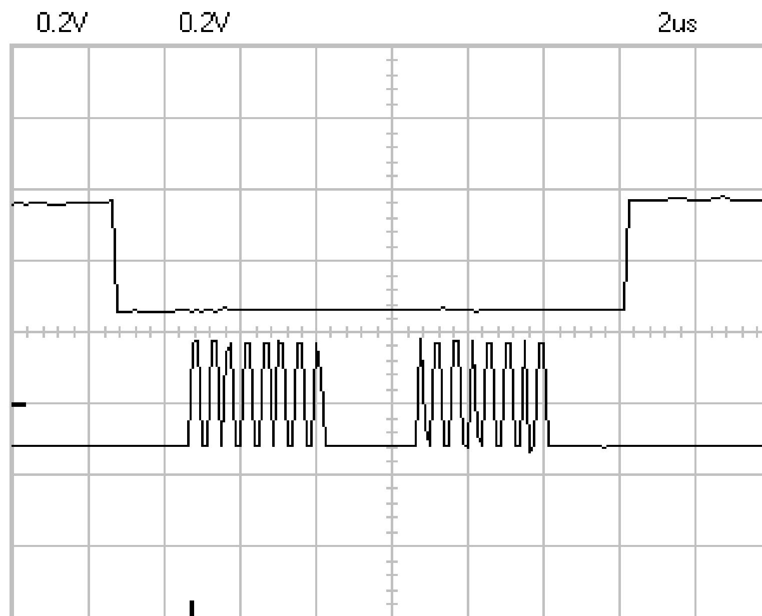


Figure 9. Probe Placed on the SPI0 Header SS Signal Line

If the interface between the SPI0 and the TLC2543 is correct, the ADC output data should be seen as shown in the upper signal in **Figure 10**. This figure shows the ADC output data signal obtained by applying the second oscilloscope probe to the SPI0's MISO port. The exact appearance of this signal is highly dependent on the nature of the data being digitized by the ADC on the prototype board; in contrast, the signals' appearances in Figure 10, Figure 11, and Figure 12 are fairly static in appearance over the period of the run.

If output data does not appear as expected, check all connections. Make sure that the ground systems are securely connected on all subsystems if noise occurs. This advice applies to all circuits discussed in this paper.

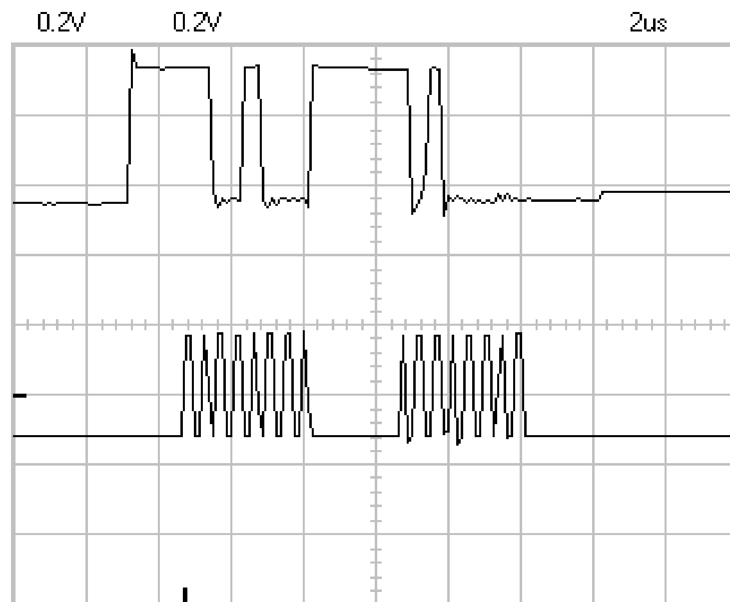


Figure 10. ADC Data

4. Integrating Devices With the EVM Board

Earlier sections of this note established the ease of interfacing to various devices, such as an LCD/keypad display and an 11-channel, 12-bit analog-to-digital converter. Remember that in the descriptions of interfacing the MPX4115A with the TLC2543 and the DSP, the barometric measurements were collected, but the debugger was required to see the results. By combining the LCD/keypad and the ADC located on the DSP's SPI0 port, it is now possible to show, in nearly real-time, the measurements collected and processed on the LCD display, using the code shown in **Code Example 3**. This code, *IntFaceBaro.c*, shows use of both LCD/keypad and TLC2543 ADC chip, interfaced to an atmospheric transducer.

Code Example 3. *IntFaceBaro.c*

```
/* IntFaceBaro.c - Procedure used to interface to TLC2543 - 12-bit analog-
   to-digital converter with serial control and 11 analog inputs. The
   DSP56824 SPI0 port is employed. The software is developed for the
   DSP56824 EVM environment. Jan 26, 2001 */
```

```
#include "io.h"
#include "fcntl.h"
#include "bsp.h"
#include "spi.h"
#include "stdio.h"
#include "serial.h"
#include "string.h"
#include "port.h"
#include "timer.h"
#include "types.h"
#include "math.h"
#include "lcd.h"
#include "TLC2543.h"

void BaroAdj( int , Word32 * );


void main(void)
{

   spi_sParams      SpiParams;

   int              Uart;
   UWord16          NewUartState;
   char             TitleString1[] = {"DSP Interface Demo"};
   char             TitleString2[] = {"  Barometric Meas.\n"};
   char             astring[20];
   static char      inputarrry[8];

   struct timespec TenthSecond = {0, 100000000};
   struct timespec OneMillisecond = {0,1000000};

   int              SerialMaster;
   static UWord16          ADcmd;
   static UWord16   Datain, DataStore[70];
   Word16           i, ii;
   static Word32    sum;
   static Word32    Vcount;


  static Word32    BaroPress, BAdj;


   SpiParams.bSetAsMaster = 1;    /* SPI0 is set as master */

   SerialMaster = open( BSP_DEVICE_NAME_SPI_0, 0, &SpiParams );

   /* Open Serial Port via SPI1 and Uart */

   Uart = open( BSP_DEVICE_NAME_SERIAL_0, 0 );

   NewUartState = MAX3100_FIFO_DISABLE | \
                  MAX3100_INT_ENABLE_DATA | \
                  MAX3100_IR_DISABLE | \
                  MAX3100_STOPBIT_1 | \
                  MAX3100_PARITY_NONE | \
                  MAX3100_WORD_8BIT | \
                  MAX3100_BAUD_9600;
```

```
ioctl( Uart, SERIAL_DEVICE_RESET, &NewUartState );

/* Set bit clock rate so sampling rate */
ioctl( SerialMaster, SPI_PHI_DIVIDER_32, NULL );

/* Set Data format for 16 bit */
ioctl( SerialMaster, SPI_DATAFORMAT_RAW, NULL );

/* SS can be left low between successive SPI bytes */
ioctl( SerialMaster, SPI_CLK_PHASE_SS_CLEAR, NULL);

/* TLC2543 is commanded to use analog input 8,
   Output data length = 16 bits,
   Output data format = MSB first, unsigned   */


ADcmd = TLC2543_port_8 | \
        TLC2543_data_16 | \
        TLC2543_MSB_first | \
        TLC2543_Unipolar;


write( Uart, NewScr, 1 );
write( Uart, TitleString1, strlen(TitleString1));
write( Uart, TitleString2, strlen(TitleString2));
 /* determine if there are any user keypad entries */

PosCursor[1] = 42;  /* Set Cursor position in LCD */


write( Uart, PosCursor, 2);

write ( Uart, "Adj. for site alt\n", 18);
ii=0;
read ( Uart, &inputarray[ii], 1);

if ( strcmp(inputarray, "A") == 0 )
{
    BaroAdj( Uart, &BAdj );
    write( Uart, NewScr, 1 );
    write( Uart, TitleString1, strlen(TitleString1));
    write( Uart, TitleString2, strlen(TitleString2));
}



ii = 0;
for(  ;  ;  )
{
   sum = 0;
   for ( i = 0; i < 64; i++ )
   {
      /* send command word to TLC2543  */
      write(SerialMaster, &ADcmd, sizeof(UWord16));

      /* read input from TLC2543  */
      read(SerialMaster, (UWord16 *)&Datain, sizeof(UWord16));

      Datain = Datain >> 4;
```

```
          /* process sleeps for 1 msec. */
          nanosleep( &OneMillisecond, NULL );

          DataStore[i] = Datain;
          sum += Datain;

       }
       Vcount = (sum >> 6);

//     BaroPress = (Vcount >> 2) + (Vcount >> 6) + (Vcount >> 8)
//                    + 105;

       /* Compute Barometric pressure in millibars, the following
          expression uses 32-bit arithmetic and 10 bits left shift on
          all coefficients, equation scaled for millibars and a
          12 bit ADC is
                   P = 0.271267*Vcount + 105.56
          Original equation taken from MPX4115A Tech Data sheet.  */

       BaroPress = (278 * Vcount + 108093) >> 10;

       BaroPress += BAdj;


      PosCursor[1] = 42;  /* Set Cursor position in LCD */
       write( Uart, PosCursor, 2);

       /* Send result to LCD  */
       sprintf( astring, "P = %ld mbars", BaroPress );
       write( Uart, astring, strlen(astring) );
   }

   close(SerialMaster);

}

/* This procedure is called when it is determined that the user desires
   to enter a correction to the barometric pressure measurement to
   compensate for the altitude above sea-level of the sensor.
   It is expected that the user will provide the pressure adjustment in
   Millibars.  */

void BaroAdj( int Uart, Word32 *padj )
{
    char astring[]={"Baro Height Adj\n"};
    char bstring[]={"Enter (mb) = "};
    char iarray[8];
    int  icount, temp, i, tenpow;

    write( Uart, NewScr, 1 );
    PosCursor[1] = 22;
    write( Uart, PosCursor, 2);
    write( Uart, astring, strlen(astring));
    PosCursor[1] = 42;
    write( Uart, PosCursor, 2);
    write( Uart, bstring, strlen(bstring));
```

```
        icount = 0;
        while (true)
        {
           read( Uart, &iarray[icount], 1 );
           if ( iarray[icount] == CR[0] ) break;
           if ( (iarray[icount] >= '0') && (iarray[icount] < '9'))
           {
               write( Uart, &iarray[icount], 1 );
               icount++;
           }

        }

        iarray[icount] = 0;
        sscanf(iarray, "%d", &temp );

        *padj = (Word32)temp;
    }
```

The *IntFaceBaro.c* code's header files includes two new header files: *lcd.h* and *TLC2543.h,* which provide the defines necessary to ease the use of the LCD/keypad and the TLC2543 ADC chip. The new header files are shown in **Appendix C.**

Set up the serial and SPI0 ports as in earlier examples. A message is displayed on the LCD to indicate the demo, and asks the user for input. In this case, to insert a pressure adjustment, input an "A" and "E" (=CR); for keypad key assignment values, see **Figure 4**. Another screen is generated when the procedure *BaroAdj()* requests the pressure adjustment; *BaroAdj()* works only for positions at or above sea level. The adjustment value is added to the measured value. **Appendix D.** provides information on determining this barometric pressure adjustment. If the user selects any other key, the procedure displays the barometric pressure from the MAX4115A without adjustment.

The barometric pressure results appear on the display. Since the barometric pressure changes rather slowly, you can show a rapid change of pressure by using a straw to blow into the tube-like extension that projects perpendicular to the MAX4115A cylindrical body. While blowing, you should see the barometric pressure value change on the LCD. Using the straw to blow across the pressure device's tube should decrease the pressure, an example of theVenturi Effect.

Appendix A. Details of NewMedia's Serial LCD+

The Serial LCD+ is a 4x20 LCD display with a built-in bi-directional serial interface. The unit is controlled using standard RS-232 serial signals from a host computer or micro-controller. The LCD+ supports the following serial data rates: 1200, 2400, 4800, 9600, 19200, 38400, and 57600 baud.

A.1 Pin Definitions

Figure A-1 shows a top down of the LCD+ (the view with the LCD display module removed). The pin definitions are denoted.

A.2 Serial I/O

The serial I/O header is made up of five connections; **Table A-1** defines the pins.

Table A-1. Serial I/O pin definitions

Pin #	Description
1	TX serial output - nearest to top in Figure 1
2	RX serial input – next pin below TX
3	Not connected
4	GND – connected to host comp/µc serial ground
5	A courtesy +5V courtesy connection (max 20 mA)

A.3 Power Input

The power input section consists of four through holes (Solder pads). The two holes marked GND are grounds. The hole marked +5.5V to +15V is tied to the LCD+ module's onboard regulator. The hole marked +5V ties to the +5V buss and is used to bypass the LCD+ onboard regulator when a regulated +5V source is supplied.

A.4 ADC Inputs

The eight ADC inputs are labeled 1 – 8. By default, all ADC inputs are set to read voltage in the 0 to +5V range.

A.5 Relay Driver Outputs

There are nine Relay Driver connections, labeled on the underside of the board as R1-8 and RLY_VDC. The connections labeled R1-8 are the relay driver chip outputs; the RLY+VDC connection provides access to the ULN2803A driver chip's internal back EMF protection diodes.

A.6 Matrix KeyPad Input

The keypad input connections (visible in **Figure A-1**) are the upper-most eight of the 8x2 header connection. The lower eight of the 8x2 header connection (not visible in **Figure A-1**) are used by the factory for programming and should remain unconnected.

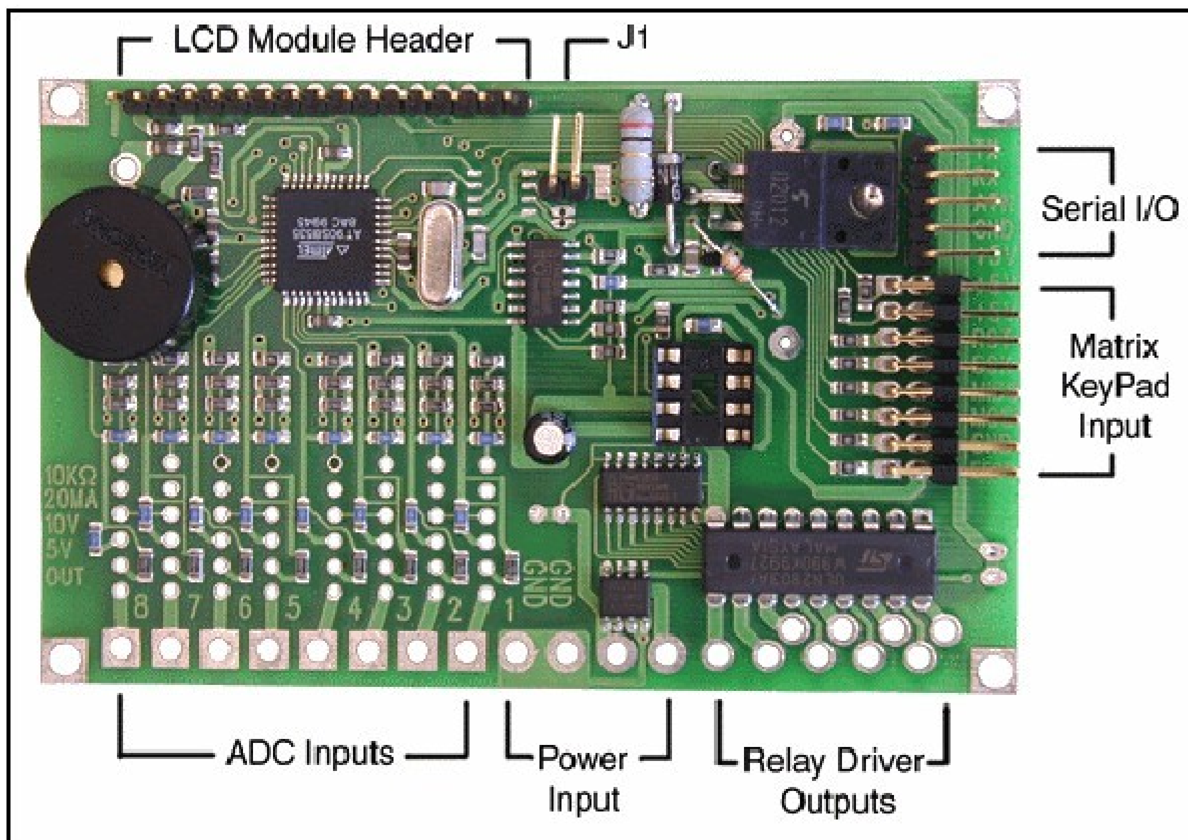


Figure A-1. LCD+ Pin-Outs

A.7 Interfacing the LCD+

The LCD+ can be controlled using any computer or microprocessor supporting 1200-57600 baud data rates with a 8,N,1 data format (**8** data bits, **N**o parity, **1** stop bit).

A.8 Keypad Interface

The keypad interface supports matrix keypads up to 4x4 in size (16 keys). Instead of predefining the keypad key serial data format as 0 through 15, each of the keypad's keys is serially represented by a user-definable byte value. This user-definable value (Tag) is stored within the LCD+ EEPROM as a 0-15 byte array. Each byte of the array corresponds to a key on the keypad (i.e. key 0 corresponds to byte 0 of the array). Whenever a key is pressed, the stored byte representation for that key number is sent serially.

A.9 Keypad Options

Various keypad options are supported by six user-definable options or modes. To set these modes, send a CTRL-X, followed by your command byte containing the desired modes. As shown in **Table A-2**, placing a 1 in any of the bits turns its corresponding option on, and a 0 turns it off. Control Codes for the LCD+ are shown in **Table A-3**.

Table A-2. LCD+ Keypad Options Table

Mode Byte	Name	Description
B0	"Key Beeps"	Beep buzzer during each key press
B1	"Key Press Format"	Send one byte for key down and one for key up
B2	"LCD Echo"	Echo key press data ASCII representation to LCD display
B3	"Mask Key Presses"	Display all key presses as asterisks on LCD
B4	"Auto Backlight"	Turns on backlight with any key press and off 4 seconds after last key press
B7	"Delayed Response"	Provide 3.0ms delay in response to a command. Required for interfacing with BS2

Table A-3. LCD+ Control Codes

Control Code	Function	Total bytes needed + Command_Data	Return Data
Ctrl-A	Cursor Home	1 Byte(0x01)	None
Ctrl-B	Set/Adjust Backlight Brightness	2 Bytes (0x02) + 0-255	None
Ctrl-C	Set/Adjust Contrast	2 Bytes (0x03) + 0-255	None
Ctrl-D	Hide Cursor	1 Byte (0x04)	None
Ctrl-E	Underline Cursor	1 Byte (0x05)	None
Ctrl-F	Block Cursor	1 Byte (0x06)	None
Ctrl-G	Sound Bell/Buzzer	1 Byte (0x07)	None
Ctrl-H	Backspace	1 Byte (0x08)	None
Ctrl-I	Horizontal Tab	1 Byte (0x09)	None
Ctrl-J	Line Feed	1 Byte (0x0a)	None
Ctrl-K	Reverse Line Feed	1 Byte (0x0b)	None
Ctrl-L	Form Feed/Clear Screen	1 Byte (0x0c)	None
Ctrl-M	Carriage Return	1 Byte (0x0d)	None
Ctrl-N	Backlight On	1 Byte (0x0e)	None
Ctrl-O	Backlight Off	1 Byte (0x0f)	None
Ctrl-P	Set Cursor Position	2 Bytes (0x10) + 0-79	None
Ctrl-Q	Clear Column	1 Bytes (0x11)	None
Ctrl-R	Set Relays	2 Bytes (0x12) + 0-255	None
Ctrl-S	Define Custom Character	10 Bytes (0x13) + 0-7 + 8 Bytes	None
Ctrl-T	Download Keypad Tags	17 Bytes (0x14) + 16 New keys	None

Ctrl-U	Set Baud Rate	2 Bytes (0x15) + 0-6	None
Ctrl-V	Read ADC Inputs	2 Bytes (0x16) + 1-8	2 Bytes
Ctrl-W	Change Bell/Buzzer Frequency	2 Bytes (0x17) + 0-255	None
Ctrl-X	Set Keypad Modes	2 Bytes (0x18) + 0-255	None
Ctrl-Y	Read Keypad Input as Port	1 Byte (0x19)	1 Byte
Ctrl-Z	Get LCD+ EEPROM Settings	1 Byte (0x20)	20 Bytes
None	Display Custom Character	1 Byte (0x80)	None

Appendix B. TI's TLC2543

The following data is taken from TI's TLC2543 data sheet. It is included only to provide information that the reader may need to understand the code described in this note.

The TLC2543 is a 12-bit switched-capacitor, successive approximation, ADC converter. Each device has three control inputs (chip select (CS), the I/O clock and the address input (DATA INPUT)) and is designed for communication with the serial port of a host processor or peripheral through a serial 3-state output.

The device has an on-chip 14-channel multiplexer that can select any of 11 inputs or any of three internal self-test voltages. The sample-and-hold function is automatic. At the end-of-conversion (EOC), output goes high to indicate that conversion is complete. A switched-capacitor design allows low-error conversion over the full operating temperature range.**Figure B-1** shows the pin-outs of the 20- pin DIP version of the chip used in this note.

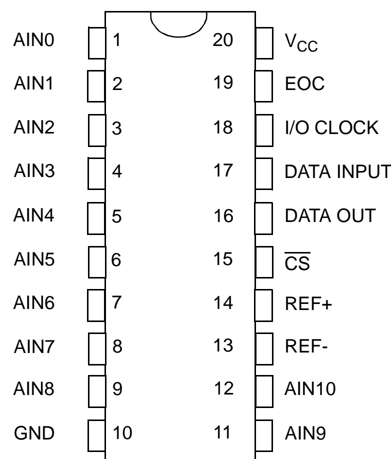


Figure B-1. TLC2543 DIP Version Pin-Outs

B.1 Operating Principles

If the $\overline{CS}$ is high, the I/O CLOCK and DATA INPUT are disabled and DATA OUT is in a high-impedance state. When the $\overline{CS}$ goes low, the conversion sequence begins by enabling the I/O CLOCK and DATA INPUT and removes the DATA OUT from the high-impedance state.

The data input is 8 bits long and is primarily a command item exercising control over the features of the chip. The following table describes the fields of the data input.

Table B-1. Fields of Data Input

Field	Description
D7-D4	Analog Channel Address
D3-D2	Data Length
D1	Output MSB or LSB First
D0	Unipolar or Bipolar Output

The I/O CLOCK sequence applied to the I/O CLOCK terminal transfers this data to the input data register.

The I/O CLOCK during this transfer shifts the previous conversion result from the output data register to DATA OUT. I/O CLOCK receives the input sequence of 8, 12, or 16 clock cycles long depending on the data-length selection in the input data register. Sampling of the analog input begins on the fourth falling edge of the input I/O CLOCK sequence and is held after the last falling edge of the I/O CLOCK sequence. The last falling edge of the I/O CLOCK sequence also takes EOC low and begins the conversion.

B.2 Data Input

The data input is internally connected to an 8-bit serial-input address and control register. The register defines the operation of the converter and the output data length. The host provides the data word with MSB first. Each data bit is clocked in on the rising edge of the I/O CLOCK sequence. **Table B-2** provides the details for the data input-register format.

Table B-2. Input-Register Format

FUNCTION SELECT	INPUT DATA BYTE							
	ADDRESS BITS				L1	L0	LSBF	BIP
	D7 (MSB)	D6	D5	D4	D3	D2	D1	D0 (LSB)
Select Input Channel								
AIN0	0	0	0	0				
AIN1	0	0	0	1				
AIN2	0	0	1	0				
AIN3	0	0	1	1				
AIN4	0	1	0	0				
AIN5	0	1	0	1				
AIN6	0	1	1	0				

Table B-2. Input-Register Format

AIN7	0 1	1 1			
AIN8	1 0	0 0			
AIN9	1 0	0 1			
AIN10	1 0	1 0			
Select Test Voltage					
$(V_{ref+} - V_{ref-})/2$	1 0	1 1			
V_{ref-}	1 1	0 0			
V_{ref+}	1 1	0 1			
Software Power Down	1 1	1 0			
Output Data Length					
8 bits			0 1		
12 bits			X 0		
16 bits			1 1		
Output Data Format					
MSB first				0	
LSB first (LSBF)				1	
Unipolar (binary)					0
Bipolar (BIP) Two's Complement					1

X = don't care

Appendix C. Header Files *TLC2543.h* and *lcd.h*

```
/*  lcd.h    some common material used with NewMedia LCD device */

   UWord16    NewScr[2]= {12,0};
   UWord16    BackLightOn[2] = {14,0};
   UWord16    LightLevel[2] = {2,70};
   UWord16    DispContrast[2] = {3,100};
   UWord16    DispKeypad[2] = {24,1};
   char       LF[]={10,0}, CR[]={'E'};
   UWord16    BS[]={8,0};
   UWord16    PosCursor[]={16,0};


/*  TLC2543.h    include file to be used with the 11 port 12-bit ADC
                  chip - TI's TLC2543   */

#define TLC2543_port_0  0x0000
#define TLC2543_port_1  0x1000
#define TLC2543_port_2  0x2000
#define TLC2543_port_3  0x3000
#define TLC2543_port_4  0x4000
#define TLC2543_port_5  0x5000
#define TLC2543_port_6  0x6000
#define TLC2543_port_7  0x7000
#define TLC2543_port_8  0x8000
#define TLC2543_port_9  0x9000
#define TLC2543_port_10 0xA000

#define TLC2543_data_8  0x0400
#define TLC2543_data_12 0x0000
#define TLC2543_data_16 0x0C00

#define TLC2543_MSB_first 0x0000
#define TLC2543_LSB_first 0x0200

#define TLC2543_Unipolar  0x0000
#define TLC2543_Bipolar   0x0001
```

Appendix D. Determining the Pressure Transducer's Altitude Pressure Adjustment

This section discusses the apparent error seen when using the MPX4115A Integrated Pressure Sensor. The units of barometric pressure used here are in metric units (SI) millibars. If the demo is performed at or near sea level, the displayed result should be identical to a reading provided by the local weather service via either the internet or radio weather services. As a standard, all barometric pressure values recorded and provided by the weather service are always referenced to sea level, regardless of the altitude of the weather measuring station.

When the demo is performed at a location above sea level, a divergence takes place between the demo-measured value and the current weather service value. The demo value will always be less and will decrease as the height above sea level increases. Under these circumstances, the MPX4115A is performing nothing more than an altimeter function. A mathematical relationship exists that relates the height above sea level to the pressure measurement.

For any given location, there is a means of obtaining the correction for the altitude. One way is to contact the local office of the weather service via its internet web site and observe the barometric pressure readings provided. A second means requires listening to the local radio outlet of the weather service (usually found around 162.0 MHz VHF using a NBFM radio) for the current barometric value. With the current values, a correction term (Δ) can be determined by the following expression:

$$\Delta = P_{weather_service} - P_{measured}$$

The value (D) is a constant for a given location and it is added to the measured barometric value as follows:

$$P_{sea\text{-}level} = P_{measure} + \Delta$$

The station pressure varies with altitude above sea level in accordance with the following expression:

$$P_{station} = P_{sea\text{-}level} * \exp(-z/H)$$

where

$H = 7000$

z = height above sea-level (meters)

For example, a local radio station announces the barometric or sea level pressure as 1013 millibars (or hectoPascals) and you live in the mountains at 1760 meters. Using the above equation, your station pressure is 788 millibars. The Δ correction in this case is 225 millibars.

Appendix E. Motorola's MPX4115A Integrated Pressure Sensor

The following is taken from the MPX4115A Integrated Pressure Sensor Data sheet.

The Motorola MPX4115A series Manifold Absolute Pressure (MAP) sensor for engine control is designed to sense absolute air pressure. Its operating pressure range is 15 to 115 kPa (2.2 to 16.7 psi) which produces a voltage output of 0.2 to 4.8V.

Motorola's MAP sensor, integrated on-chip, bipolar op-amp circuitry and thin film resistor network to provide a high-output signal and temperature compensation. The small form factor and high reliability of on-chip integration make the Motorola MAP sensor a logical and economical choice for the automotive system designers.

Among its features:

- 1.5% maximum error over 0° to 85°C
- Ideally suited for microprocessor and microcontroller-based systems
- Temperature compensated from -40° to $+125^{o}$C

Application examples:

- Aviation altimeters
- Industrial controls
- Engine control
- Weather stations and weather reporting devices

Figure E-1 shows the sensor output signal relative to pressure input. Typical minimum and maximum output curves are shown for operation over 0 to 85°C temperature range. Output will saturate outside of the rated pressure range. **Figure E-2** depicts the version of the sensor employed with this demo and also shows the pins to aid in proper use and installation.

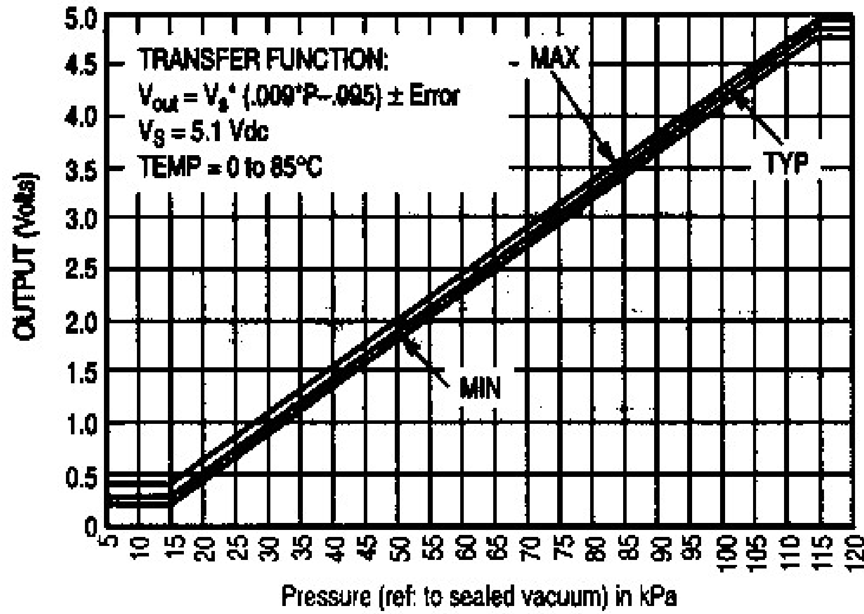
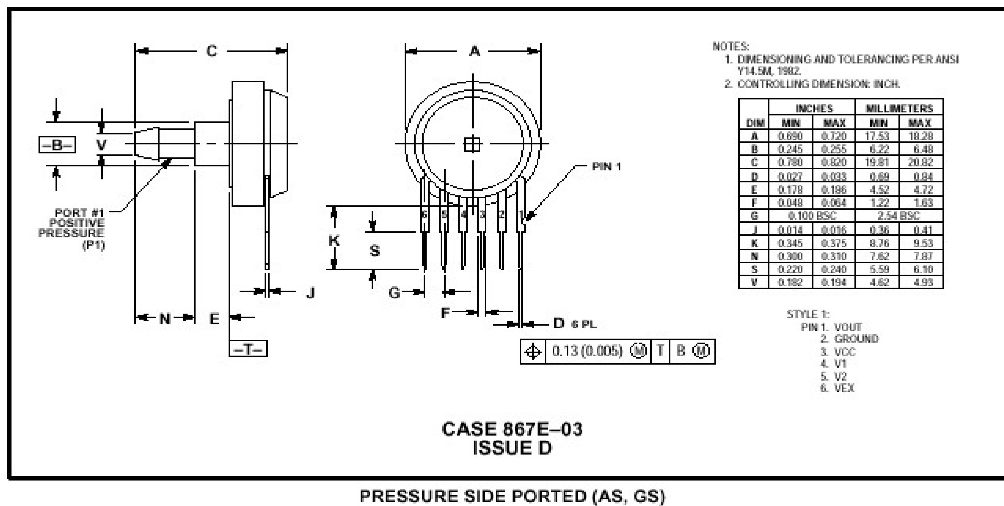


Figure E-1. Output vs. Absolute Pressure

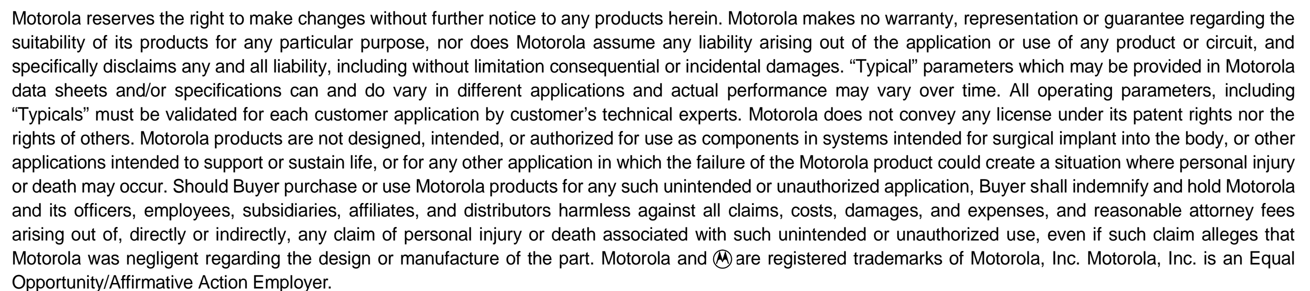


DIM	INCHES MIN	INCHES MAX	MILLIMETERS MIN	MILLIMETERS MAX
A	0.690	0.720	17.53	18.28
B	0.245	0.255	6.22	6.48
C	0.780	0.820	19.81	20.82
D	0.027	0.033	0.69	0.84
E	0.178	0.186	4.52	4.72
F	0.048	0.064	1.22	1.63
G	0.100 BSC		2.54 BSC	
J	0.014	0.016	0.36	0.41
K	0.345	0.375	8.76	9.53
N	0.300	0.310	7.62	7.87
S	0.220	0.240	5.59	6.10
V	0.182	0.194	4.62	4.93

Figure E-2. MPX4115A and Pin-Outs

NOTES:

OnCE^{TM} is a registered trademark of Motorola, Inc.


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 which may be provided in Motorola data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. 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.

How to reach us:
USA/EUROPE/Locations Not Listed: Motorola Literature Distribution: P.O. Box 5405, Denver, Colorado 80217.
1-303-675-2140 or 1-800-441-2447

JAPAN: Motorola Japan Ltd.; SPS, Technical Information Center, 3-20-1 Minami-Azabu. Minato-ku, Tokyo 106-8573 Japan.
81-3-3440-3569

ASIA/PACIFIC: Motorola Semiconductors H.K. Ltd.; Silicon Harbour Centre, 2 Dai King Street, Tai Po Industrial Estate, Tao Po, N.T., Hong Kong. 852-26668334

Technical Information Center: 1-800-521-6274

HOME PAGE: http://motorola.com/semiconductors/dsp MOTOROLA HOME PAGE: http://motorola.com/semiconductors/

MOTOROLA