Application Note 165
Interfacing the DS2760 1-Wire High Precision Li-Ion Battery Monitor in a Microcontroller Environment

www.maxim-ic.com

INTRODUCTION

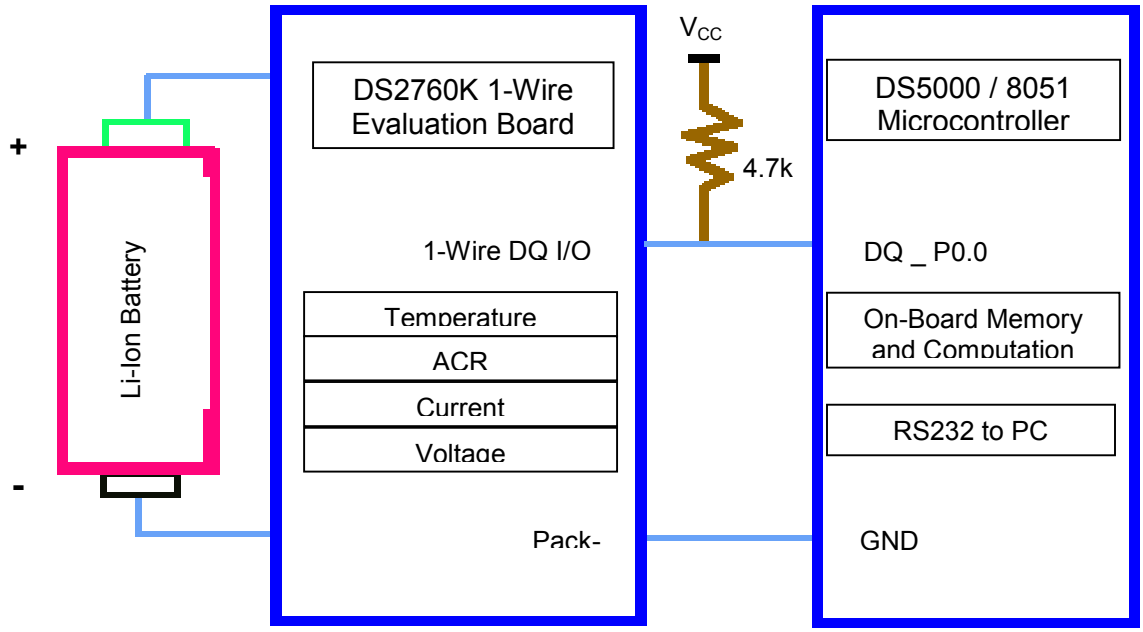
Interfacing a 1-Wire® device to a microcontroller has in the past proven to be quite challenging. Various methods have been used facilitate the process, from using a VHDL 1-Wire Master Controller, to using serial interface chips like the DS2480 and DS9123. In this application note, "C" software will be introduced which bypasses these extraneous methods to provide the user with the simplest possible single-wire solution for embedded controller applications.

In the following example, we will demonstrate a method for connecting one wire from a microcontroller to the DS2760K evaluation board to read the temperature, battery capacity (ACR), current and the voltage registers. The DS2760K hardware is used to simplify communications because the module contains the sense resistor and all the necessary circuitry to verify the accuracy of our source code using the PC kit version.

HARDWARE CONFIGURATION

The following block diagram illustrates the simplicity of the hardware configuration. The host microcontroller uses a single wire to connect to the DQ input/output pin of the DS2760 1-Wire device. In a real-world scenario, the microcontroller would contain up to 1MB of off-chip memory for cell phone or PDA applications. However, we seek here to demonstrate the fundamentals of the interface design as well as provide source code to significantly shorten the design cycle. The DS5000/DS2250 is used because of it has on-board RAM and program loader. Other micros could very easily be substituted, once timing considerations have been accounted for. A programmable delay line might be needed for some of the faster microprocessors.

DS2760K — MICROCONTROLLER INTERFACE Figure 1.0



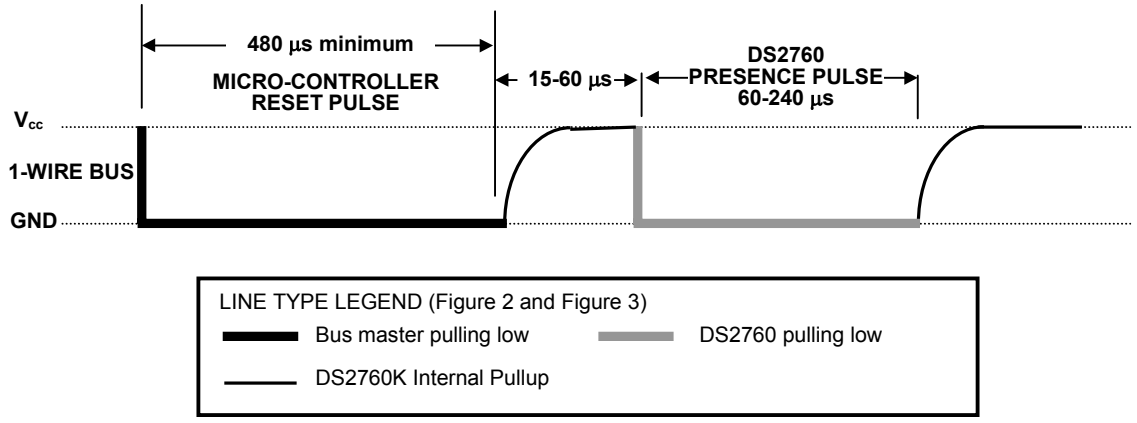
INTERFACE TIMING

A write time slot is initiated when the bus master pulls the 1-Wire bus from logic high (inactive) to logic low. There are two different characteristics for the write "0" time slot and the write "1" time slot. All write time slots must be 60µs to 120µs in duration with a 1µs minimum recovery time between cycles. During the write "0" time slot, the host microcontroller will pull the line low for the duration of the time slot. However, during the write "1" time slot, the microcontroller will pull the line low and then release the line so that the DS2760 may pull the line high within 15µs after the start of the time slot.

A read time slot is initiated when the microcontroller pulls the line low (active). The line must be kept low for 1µs and then released so that the DS2760 can take control of the line and present valid data (high or low). All read time slots must be 60µs to 120µs in duration with a minimum 1µs recovery time between cycles.

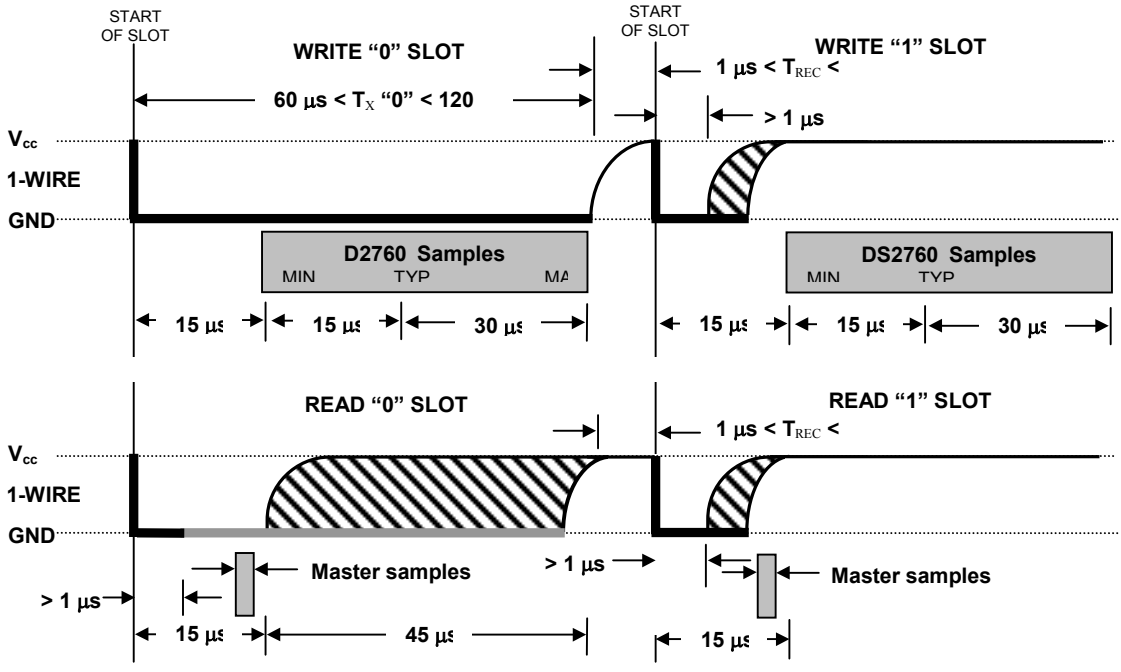
Each cycle must begin with a reset pulse to start the read or write cycle. The interface timing is illustrated in Figure 2.0 and Figure 3.0.

RESET PULSE AND PRESENCE PULSE Figure 2.0



WRITE AND READ TIME SLOTS

Figure 3.0



SOFTWARE CONTROL

In order to accurately control the special timing requirements the 1-Wire interface, certain key functions must first be established. The first function created must be the "delay" function which is integral to all read and write control. This function will be entirely dependent the speed of the microcontroller. For testing these functions, we used the DS5000 (8051-compatible) microcontroller running at 11.059MHz. The following example illustrates the "C" prototype function for creating the timing delay.

```
// DELAY - with an 11.059MHz crystal.
//  Calling the routine takes about 24us, and then
//  each count takes another 16us.
//
void delay(int useconds)
{
        int s;
        for (s=0; s<useconds;s++);
}
```

Since each cycle must begin with a reset function, the "reset" prototype function is the next most important function to be implemented. The reset time slot is 480µs. By setting a delay of "3", followed by "25", we are effectively setting our reset pulse (low) and then allowing the DS2760 to pull the line low to indicate the device "presence."

```
////////////////////////////////////////////////////////////////////////////
// OW_RESET - performs a reset on the one-wire bus and
//   returns the presence detect. Reset is 480us, so delay
//   value is (480-24)/16 = 28.5 - we use 29. Presence checked
//   another 70us later, so delay is (70-24)/16 = 2.875 - we use 3.
//
unsigned char ow_reset(void)
{
      unsigned char presence;

          DQ = 0;                         //pull DQ line low
          delay(29);                      // leave it low for 480us
          DQ = 1;                   // allow line to return high
          delay(3);                       // wait for presence
          presence = DQ;                  // get presence signal
          delay(25);                      // wait for end of timeslot
          return(presence);         // presence signal returned
}                   //    0=presence, 1 = no part
```

The following read and write prototype functions provide the backbone of all data bit and data byte read and write operations.

```
//////////////////////////////////////////////////////////////////////////
// READ_BIT - reads a bit from the one-wire bus. The delay
//   required for a read is 15us, so the DELAY routine won't work.
//   We put our own delay function in this routine in the form of a
//   for() loop.
//
unsigned char read_bit(void)
{
        unsigned char i;

        DQ = 0;                         // pull DQ low to start timeslot
        DQ = 1;                         // then return high
        for (i=0; i<3; i++);        // delay 15us from start of timeslot

        return(DQ);                     // return value of DQ line
}
```

```
//////////////////////////////////////////////////////////////////////////
// WRITE_BIT - writes a bit to the one-wire bus, passed in bitval.
//
void write_bit(char bitval)
{
        DQ = 0;                         // pull DQ low to start timeslot
        if(bitval==1) DQ =1;        // return DQ high if write 1
        delay(5);                               // hold value for remainder of timeslot
        DQ = 1;

}// Delay provides 16us per loop, plus 24us. Therefore delay(5) = 104us
```

```
//////////////////////////////////////////////////////////////////////////
// READ_BYTE - reads a byte from the one-wire bus.
//
unsigned char read_byte(void)
{
        unsigned char i;
        unsigned char value = 0;

        for (i=0;i<8;i++)
        {
                if(read_bit()) value|=0x01<<i;        // reads byte in, one byte at a time and then
                                                                                //           shifts it left
                delay(6);                                                       // wait for rest of timeslot
        }
        return(value);
}
```

```
////////////////////////////////////////////////////////////////////////
// WRITE_BYTE - writes a byte to the one-wire bus.
//
void write_byte(char val)
{
        unsigned char i;
        unsigned char temp;

        for (i=0; i<8; i++)                  // writes byte, one bit at a time
        {
                temp = val>>i;                          // shifts val right 'i' spaces
                temp &= 0x01;                // copy that bit to temp
                write_bit(temp);             // write bit in temp into
        }
        delay(5);

}
```

READING DEVICE TEMPERATURE

If there is a single device on the net, then the "Read Temperature" function can be used directly as shown below.

```
void Read_Temperature(void)
{
unsigned short get[3];
float temp_lsb,temp_msb;
int k;
float temp_f,temp_c;

unsigned short temp_sign,sign;

                        ow_reset();
                        write_byte(0xCC); // Skip ROM
                        write_byte(0x69);  // Read Temp. Registers
                        write_byte(0x18);
                        for (k=0;k<2;k++){get[k]=read_byte();}// Get the data

                        temp_msb = get[0]; // Sign bit + msbbyte
                        temp_lsb = (get[1] & 0xE0) >>4; // data lsbyte, mask all but upper three bits
                        printf("\nTemperature Register MSB = %X   |   ",get[0]);
                        printf("  Temperature Register LSB = %X\n",get[1]);
                        temp_sign = temp_msb;  // check sign bit
                         sign = temp_sign & 0x80; // mask all but the sign bit
                        if (sign != 0) {(unsigned short)temp_sign = ((~(unsigned short)temp_msb) + 1) &0xFF;}
                        // twos complement
                        if (sign != 0) {temp_msb = ((-1)*(float)temp_sign);} // add sign bit
                        temp_c =  (float)temp_msb + temp_lsb/16;
                        printf("\n  TempC = %5.3f degrees C\n",temp_c);  // print temp. C
                        temp_f = ((temp_c)* 9)/5 + 32;
                        printf("\n  TempF=  %5.3f degrees F\n",temp_f);  // print temp. F
}
```

The “Read ROM” command is used to find the 64-bit ROM code when only a single device is on the net. Multiple devices require the use of the “Search ROM” functions not shown here. The “ACR”, “Current”, and “Voltage” prototype functions are shown below.

```
void Read_ROMCode(void)
{
int n;
char dat[9];
                printf("\nReading ROM Code\n");
                ow_reset();
                write_byte(0x33);
                for (n=0;n<8;n++){dat[n]=read_byte();}


printf("\n ROM Code = %X%X%X%X\n",dat[7],dat[6],dat[5],dat[4],dat[3],dat[2],dat[1],dat[0]);
}
```

```
void Read_ACR(void)
{
unsigned short get[3];
float ACR_lsb,ACR_msb,ACR;
int k;

unsigned short ACR_sign,sign;

        ow_reset();
        write_byte(0xCC); // Skip ROM
        write_byte(0x69);  // Read ACR Registers
        write_byte(0x10);

for (k=0;k<2;k++){get[k]=read_byte();}// Get the data

        ACR_msb = get[0]; // Sign bit + msbbyte
        ACR_lsb = get[1]; // data lsbyte
        printf("\nACR Register MSB = %X   |   ",get[0]); //print MSB
        printf(" ACR Register LSB = %X\n",get[1]); //print LSB
        ACR_sign = ACR_msb;  // check sign bit
        sign = ACR_sign & 0x80; // mask all but the sign bit

if (sign != 0) {(unsigned short)ACR_sign = ((~(unsigned short)ACR_msb) + 1) &0xFF;} // twos
complement

if (sign != 0) {ACR_msb = ((-1)*(float)ACR_sign);} // add sign bit
ACR = ((((float)ACR_msb)*256 + ACR_lsb)*0.00625)/0.025;
// make conversion, then divide by sense resistor

printf("\n  ACR = %5.4f mA-Hrs\n",ACR);  // print ACR

}
```

```
void Read_Current(void)
{
unsigned short get[3];
float Current_lsb,Current_msb,Current;
int k;

unsigned short Current_sign,sign;

        ow_reset();
        write_byte(0xCC); // Skip ROM
        write_byte(0x69);  // Read Voltage Registers
        write_byte(0x0E);
for (k=0;k<2;k++){get[k]=read_byte();}// Get the data

        Current_msb = get[0]; // Sign bit + msbbyte
        Current_lsb = get[1]&0xE0; // data lsbyte
        printf("\nCurrent Register MSB = %X   |   ",get[0]); //print MSB
        printf(" Current Register LSB = %X\n",get[1]); //print LSB
        Current_sign = Current_msb;  // check sign bit
        sign = Current_sign & 0x80; // mask all but the sign bit
if (sign != 0) {(unsigned short)Current_sign = ((~(unsigned short)Current_msb) + 1) &0xFF;} // twos
complement
if (sign != 0) {Current_msb = ((-1)*(float)Current_sign);} // add sign bit
Current = (((float)Current_msb*256)/8)*0.000015625 + (((float)Current_lsb/16)/8)*0.000015625; // make
conversion;
        Current = Current/0.25; // divide by sense resisor
        printf("\n  Current = %5.3f mA\n",Current);  // print Current

}
```

```
void Read_Voltage(void)
{
unsigned short get[3];
float Voltage_lsb,Voltage_msb,Voltage;
int k;

unsigned short Voltage_sign,sign;

        ow_reset();
        write_byte(0xCC); // Skip ROM
        write_byte(0x69);  // Read Voltage Registers
        write_byte(0x0C);
for (k=0;k<2;k++){get[k]=read_byte();}// Get the data

        Voltage_msb = get[0]; // Sign bit + msbbyte
        Voltage_lsb = get[1]&0xE0; // data lsbyte
        printf("\nVoltage Register MSB = %X   |   ",get[0]);
        printf(" Voltage Register LSB = %X\n",get[1]);
        Voltage_sign = Voltage_msb;  // check sign bit
        sign = Voltage_sign & 0x80; // mask all but the sign bit
if (sign != 0) {(unsigned short)Voltage_sign = ((~(unsigned short)Voltage_msb) + 1) &0xFF;} // twos
complement

if (sign != 0) {Voltage_msb = ((-1)*(float)Voltage_sign);} // add sign bit
Voltage = (((float)Voltage_msb*256)/32)*.00488  + (Voltage_lsb/16)*0.00488; // make conversion
        printf("\n  Voltage = %5.3f Volts\n",Voltage);  // print Voltage
 }
```

```
void Read_Memory(void)
{
int j;
unsigned short pad[28];
                        printf("\nView Data Registers\n\n");
                        write_byte(0x69);
                        write_byte(0x00);
                        for (j=0;j<26;j++){pad[j]=read_byte();}
printf("\nProtection Register = %X   |   ",pad[0]);
printf(" Status Register = %X\n",pad[1]);
printf("\nEEPROM Register = %X   |   ",pad[7]);
printf(" Special Features Register = %X\n",pad[8]);
printf("\nVoltage Register MSB = %X   |   ",pad[12]);
printf(" Voltage Register LSB = %X\n",pad[13]);
printf("\nCurrent Register MSB = %X   |  ",pad[14]);
printf("  Current Register LSB = %X\n",pad[15]);
printf("\nAccumulated Current Register MSB = %X   |   ",pad[16]);
printf("  Accumulated Current Register LSB = %X \n",pad[17]);
printf("\nTemperature Register MSB = %X   |   ",pad[24]);
printf("  Temperature Register LSB = %X\n",pad[25]);
}
```

```
char       WritePort( char MSB_data1, char LSB_data2)
{
           char input_data;
           input_data=0;
           MSB_data1=0;
           LSB_data2=0;
           printf ("\nEnter ACR Hex Byte   ");
           MSB_data1 = getchar();
           MSB_data1 = Check_Hex(MSB_data1);
           LSB_data2 = getchar();
           LSB_data2 = Check_Hex(LSB_data2);
           input_data= MSB_data1&0x0F;
           input_data = input_data<<4;
           input_data= input_data|LSB_data2&0x0F;
           write_byte((short)input_data); // Data Bus
           printf ("\n\n Hex ACR data = out %X\n",(short)input_data);
   return (short)input_data;
}
```

```
char       Check_Hex(char Hex_data)
{
                                if (Hex_data==0x61) Hex_data = 0x0A;    /*  convert 'a'  */
                                if (Hex_data==0x62) Hex_data = 0x0B;    /*  convert 'b'  */
                                if (Hex_data==0x63) Hex_data = 0x0C;    /*  convert 'c'  */
                                if (Hex_data==0x64) Hex_data = 0x0D;    /*  convert 'd'  */
                                if (Hex_data==0x65) Hex_data = 0x0E;    /*  convert 'e'  */
                                if (Hex_data==0x66) Hex_data = 0x0F;    /*  convert 'f'  */
                                if (Hex_data==0x41) Hex_data = 0x0A;    /*  convert 'A'  */
                                if (Hex_data==0x42) Hex_data = 0x0B;    /*  convert 'B'  */
                                if (Hex_data==0x43) Hex_data = 0x0C;    /*  convert 'C'  */
                                if (Hex_data==0x44) Hex_data = 0x0D;    /*  convert 'D'  */
                                if (Hex_data==0x45) Hex_data = 0x0E;    /*  convert 'E'  */
                                if (Hex_data==0x46) Hex_data = 0x0F;    /*  convert 'F'  */
           return Hex_data;
}
```

APPENDIX A
DS5000 (8051 SOURCE CODE)

```
// ds2760k.c -- Functions for the Dallas Semiconductor DS2760K
// Li-Ion Battery Monitor 1-Wire Kit Module
//  Designed for 8051 microcontrollers
// This code was developed using the DS5000/DS2250-64-16
// and the Keil version 6.1 DS5000T source code Compiler
/*---------------------------------------------------------------------*/
//#pragma CODE SMALL OPTIMIZE(3)
 /* command line directives      */
#include <absacc.h>               /* absolute addressing modes    */
#include <ctype.h>                /* character types              */
#include <math.h>                  /* standard math               */
#include <stdio.h>                 /* standard I/O                */
#include <string.h>                /* string functions            */
#include "ds50001w.h"                 /* DS5000 series 8052 registers */
/*---------------------------------------------------------------------*/
/*      Configuration parameters                                       */
/*---------------------------------------------------------------------*/
#define XtalFreq       (11059490)     /* main crystal frequency       */
#define CntrFreq       (XtalFreq/12)  /* main counter frequency       */
#define BaudRate        (9600)        /* baud rate                    */
#define CntrTime        (8)           /* number of cycles for counter */
#define Ft             (32768.0)      /* target crystal frequency     */
/*---------------------------------------------------------------------*/
/*---------------------------------------------------------------------* /

////////////////////////////////////////////////////////////////////////////
// DELAY - with an 11.059MHz crystal.
//  Calling the routine takes about 24us, and then
//  each count takes another 16us.
//
void delay(int useconds)
{
          int s;
          for (s=0; s<useconds;s++);
}

////////////////////////////////////////////////////////////////////////////
// OW_RESET - performs a reset on the one-wire bus and
//   returns the presence detect. Reset is 480us, so delay
//   value is (480-24)/16 = 28.5 - we use 29. Presence checked
//   another 70us later, so delay is (70-24)/16 = 2.875 - we use 3.
//
unsigned char ow_reset(void)
{
      unsigned char presence;

          DQ = 0;                              //pull DQ line low
          delay(29);                           // leave it low for 480us
          DQ = 1;                        // allow line to return high
          delay(3);                            // wait for presence
          presence = DQ;                       // get presence signal
          delay(25);                           // wait for end of timeslot
          return(presence);              // presence signal returned
}                     //     0=presence, 1 = no part

////////////////////////////////////////////////////////////////////////////
```

```
//////////////////////////////////////////////////////////////////////////
// READ_BIT - reads a bit from the one-wire bus. The delay
//   required for a read is 15us, so the DELAY routine won't work.
//   We put our own delay function in this routine in the form of a
//   for() loop.
//
unsigned char read_bit(void)
{
        unsigned char i;
        DQ = 0;                         // pull DQ low to start timeslot
        DQ = 1;                         // then return high
        for (i=0; i<3; i++);            // delay 15us from start of timeslot
        return(DQ);                     // return value of DQ line
}
//////////////////////////////////////////////////////////////////////////
// WRITE_BIT - writes a bit to the one-wire bus, passed in bitval.
//
void write_bit(char bitval)
{
        DQ = 0;                         // pull DQ low to start timeslot
        if(bitval==1) DQ =1;        // return DQ high if write 1
        delay(5);                               // hold value for remainder of timeslot
        DQ = 1;
}
//////////////////////////////////////////////////////////////////////////
// READ_BYTE - reads a byte from the one-wire bus.
//
unsigned char read_byte(void)
{
        unsigned char i;
        unsigned char value = 0;
        for (i=0;i<8;i++)
        {
                if(read_bit()) value|=0x01<<i;        // reads byte in, one byte at a time and then
                                                                                //           shifts it left
                delay(6);                                                       // wait for rest of timeslot
        }
        return(value);
}
//////////////////////////////////////////////////////////////////////////
// WRITE_BYTE - writes a byte to the one-wire bus.
//
void write_byte(char val)
{
        unsigned char i;
        unsigned char temp;

        for (i=0; i<8; i++)             // writes byte, one bit at a time
        {
                temp = val>>i;                          // shifts val right 'i' spaces
                temp &= 0x01;                 // copy that bit to temp
                write_bit(temp);              // write bit in temp into
        }
        delay(5);
}
```

```
//////////////////////////PROTOTYPE FUNCTIONS//////////////////////////////
char       Check_Hex(char Hex_data);
char       WritePort( char MSB_data1, char LSB_data2);
char MSB_data1,LSB_data2;
void Read_Memory(void);
void Read_ROMCode(void);
void Read_Temperature(void);
void Read_Current(void);
void Read_Voltage(void);
void Read_ACR(void);
//////////////////////////BEGIN MAIN PROGRAM//////////////////////////////
main()
{
/*---------------------------------------------------------------------*/
/*      Local variables                                                */
/*---------------------------------------------------------------------*/
unsigned char   Select_Type;    /* Function variable                   */
/*---------------------------------------------------------------------*/
/*      Start of program execution                                     */
/*---------------------------------------------------------------------*/
/*      Inhibit the watchdog timer and set up memory      */
/*---------------------------------------------------------------------*/
TA      = 0xAA;                     /* timed access                    */
TA      = 0x55;
PCON    = 0x00;                   /* inhibit watchdog timer    */
/*---------------------------------------------------------------------*/
/*      Set up the serial port                                         */
/*---------------------------------------------------------------------*/
SCON    = 0x50;    /* SCON: mode 1, 8-bit UART, enable rcvr */
TMOD    = 0x21;    /* TMOD: timer 1, mode 2, 8-bit reload  */
                   /* TMOD: timer 0, mode 1, 16-bit                   */

PCON   |= 0x80;   /* SMOD = 1 Double Baud Rate for TH1 load */
TH0=TL0 = 0;
TH1=TL0 = (unsigned int)(256 - ( (XtalFreq / BaudRate) / 192));
TR0     = 1;           /* TR0:  timer 0 run                           */
TR1     = 1;           /* TR1:  timer 1 run                           */
TI      = 1;          /* TI:   set TI to send first char of UART  */
/*---------------------------------------------------------------------*/
/*      Display DS1820 One-Wire Device  banner         */
/*---------------------------------------------------------------------*/
printf ("\n");
printf ("        Dallas Semiconductor - Systems Extension\n");
printf ("        Source  for DS2760K Li-Ion Battery Monitor \n");
printf ("        Kit Module PD110800 .\n");
printf ("        Updated  Code November 20, 2001 \n");
printf ("        [C Program for DS500x/DS2250 or 8051 Compatible Microcontroller]");
printf("\n\n");
printf("\n*********************************************************************\n");
printf ("            Select Menu Option\n");
printf ("            1. One-Wire Reset\n");
printf ("            2. Read ROM Code of Single Device On Net\n");
printf ("            3. Read Register Data\n");
printf ("            4. Enter Write ACR Data [MSB,LSB]\n");
printf ("            5. Read Temperature\n");
printf ("            6. Read Current\n");
printf ("            7. Read Current Accumulator (ACR)\n");
printf ("            8. Read Voltage\n");
printf ("\n\n");
printf (" Note: This program represents an example only.\n");
printf (" No warranties or technical support is provided with this program.\n");
```

```
/*---------------------------------------------------------------------*/
do {
/*---------------------------------------------------------------------*/
/*      Enable CE2                                                     */
/*---------------------------------------------------------------------*/
EA         = 0;                          /* Inhibit interrupts          */
TA      = 0xAA;                  /* timed access              */
TA      = 0x55;
MCON  = MCON |= 0x04;                        /* Enable topside CE        0xCC     */
/*---------------------------------------------------------------------*/
/*      Disable CE2                                                    */
/*---------------------------------------------------------------------*/
TA      = 0xAA;                  /* timed access              */
TA      = 0x55;
MCON  = 0xC8;                          /* Disable topside CE          */
EA         = 1;                          /* Enable interrupts           */
Select_Type = getchar();              /*   get  variable to start   */
switch(Select_Type)
{
case '1':  printf ("\n         1. Sent 1-Wire Reset\n");
                                ow_reset();
                                break;
case '2': printf ("            2. Read ROM Code of Single Device On Net\n");
                                ow_reset();
                                write_byte(0xCC);  // Skip ROM
                                Read_ROMCode();
                                break;
case '3': printf("\n           3. Read Register Data\n ");
                                ow_reset();
                                write_byte(0xCC);  // Skip ROM
                                Read_Memory();
                                break;
case '4': printf ("\n          4. Enter Write ACR Data [MSB,LSB]\n");
                                 ow_reset();
                                 write_byte(0xCC);  // Skip ROM
                                 write_byte(0x6C);  // Write ACR Registers
                                 write_byte(0x10);  // Start Address
                                 WritePort(MSB_data1,LSB_data2); // Write a byte
                                 WritePort(MSB_data1,LSB_data2); // Write a byte
                                 break;
case '5': printf ("\n          5. Read Temperature\n");
                                 Read_Temperature();   //initiates a temperature reading
                                 break;
case '6': printf ("\n          6. Read Current\n");
                                 Read_Current();
                                 break;
case '7': printf ("\n          7. Read Current Accumulator (ACR)\n");
                                 Read_ACR();
                                 break;
case '8': printf ("\n          8. Read Voltage\n");
                                 Read_Voltage();
                                        break;
default:  printf ("\n          Typo: Select Another Menu Option\n");
                                 break;
};          /* end switch*/
} while (1);                           /* Loop forever                */
/*---------------------------------------------------------------------*/
/*      End of program                                                 */
/*---------------------------------------------------------------------*/
}
```

APPENDIX B
DS5000 (8051 C INCLUDE HEADER FILE)

```
/*---------------------------------------------------------------------------
DS5000.H

Header file for Dallas Semiconductor DS5000.
Copyright (c) 1995-1996 Keil Software, Inc.  All rights reserved.
---------------------------------------------------------------------------*/

#ifndef DS5000_HEADER_FILE
#define DS5000_HEADER_FILE 1

/*-----------------------------------------------
DS5000 Byte Registers
-----------------------------------------------*/
sfr P0   = 0x80;
sfr SP   = 0x81;
sfr DPL  = 0x82;
sfr DPH  = 0x83;
sfr PCON = 0x87;
sfr TCON = 0x88;
sfr TMOD = 0x89;
sfr TL0  = 0x8A;
sfr TL1  = 0x8B;
sfr TH0  = 0x8C;
sfr TH1  = 0x8D;
sfr P1   = 0x90;
sfr SCON = 0x98;
sfr SBUF = 0x99;
sfr P2   = 0xA0;
sfr IE   = 0xA8;
sfr P3   = 0xB0;
sfr IP   = 0xB8;
sfr MCON = 0xC6;
sfr TA   = 0xC7;
sfr PSW  = 0xD0;
sfr ACC  = 0xE0;
sfr B    = 0xF0;

/*-----------------------------------------------
DS5000 P0 Bit Registers
-----------------------------------------------*/
//sbit P0_0 = 0x80;  // Set Output Here
sbit DQ = 0x80;  // Set Output Here
sbit P0_1 = 0x81;
sbit P0_2 = 0x82;
sbit P0_3 = 0x83;
sbit P0_4 = 0x84;
sbit P0_5 = 0x85;
sbit P0_6 = 0x86;
sbit P0_7 = 0x87;
```

```
/*-------------------------------------------------
DS5000 PCON Bit Values
-------------------------------------------------*/
#define IDL_      0x01
#define STOP_     0x02
#define EWT_      0x04
#define EPFW_     0x08
#define WTR_      0x10
#define PFW_      0x20
#define POR_      0x40
#define SMOD_     0x80

/*-------------------------------------------------
DS5000 TCON Bit Registers
-------------------------------------------------*/
sbit IT0  = 0x88;
sbit IE0  = 0x89;
sbit IT1  = 0x8A;
sbit IE1  = 0x8B;
sbit TR0  = 0x8C;
sbit TF0  = 0x8D;
sbit TR1  = 0x8E;
sbit TF1  = 0x8F;

/*-------------------------------------------------
DS5000 TMOD Bit Values
-------------------------------------------------*/
#define T0_M0_            0x01
#define T0_M1_            0x02
#define T0_CT_            0x04
#define T0_GATE_          0x08
#define T1_M0_            0x10
#define T1_M1_            0x20
#define T1_CT_            0x40
#define T1_GATE_          0x80

#define T1_MASK_          0xF0
#define T0_MASK_          0x0F

/*-------------------------------------------------
DS5000 P1 Bit Registers
-------------------------------------------------*/
sbit P1_0 = 0x90;
sbit P1_1 = 0x91;
sbit P1_2 = 0x92;
sbit P1_3 = 0x93;
sbit P1_4 = 0x94;
sbit P1_5 = 0x95;
sbit P1_6 = 0x96;
sbit P1_7 = 0x97;
```

```
/*-----------------------------------------------
DS5000 SCON Bit Registers
-----------------------------------------------*/
sbit RI   = 0x98;
sbit TI   = 0x99;
sbit RB8  = 0x9A;
sbit TB8  = 0x9B;
sbit REN  = 0x9C;
sbit SM2  = 0x9D;
sbit SM1  = 0x9E;
sbit SM0  = 0x9F;

/*-----------------------------------------------
DS5000 P2 Bit Registers
-----------------------------------------------*/
sbit P2_0 = 0xA0;
sbit P2_1 = 0xA1;
sbit P2_2 = 0xA2;
sbit P2_3 = 0xA3;
sbit P2_4 = 0xA4;
sbit P2_5 = 0xA5;
sbit P2_6 = 0xA6;
sbit P2_7 = 0xA7;

/*-----------------------------------------------
DS5000 IE Bit Registers
-----------------------------------------------*/
sbit EX0  = 0xA8;
sbit ET0  = 0xA9;
sbit EX1  = 0xAA;
sbit ET1  = 0xAB;
sbit ES   = 0xAC;

sbit EA   = 0xAF;

/*-----------------------------------------------
DS5000 P3 Bit Registers (Mnemonics & Ports)
-----------------------------------------------*/
sbit RD   = 0xB7;
sbit WR   = 0xB6;
sbit T1   = 0xB5;
sbit T0   = 0xB4;
sbit INT1 = 0xB3;
sbit INT0 = 0xB2;
sbit TXD  = 0xB1;
sbit RXD  = 0xB0;

sbit P3_0 = 0xB0;
sbit P3_1 = 0xB1;
sbit P3_2 = 0xB2;
sbit P3_3 = 0xB3;
sbit P3_4 = 0xB4;
sbit P3_5 = 0xB5;
sbit P3_6 = 0xB6;
sbit P3_7 = 0xB7;
```

```
/*-----------------------------------------------
DS5000 IP Bit Registers
-----------------------------------------------*/
sbit PX0  = 0xB8;
sbit PT0  = 0xB9;
sbit PX1  = 0xBA;
sbit PT1  = 0xBB;
sbit PS   = 0xBC;

sbit RWT  = 0xBF;

/*-----------------------------------------------
DS5000 MCON Bit Values
-----------------------------------------------*/
#define SL_        0x01
#define PAA_       0x02
#define ECE2_      0x04
#define RA32_8_    0x08
#define PA0_       0x10
#define PA1_       0x20
#define PA2_       0x40
#define PA3_       0x80

/*-----------------------------------------------
DS5000 PSW Bit Registers
-----------------------------------------------*/
sbit P    = 0xD0;

sbit OV   = 0xD2;
sbit RS0  = 0xD3;
sbit RS1  = 0xD4;
sbit F0   = 0xD5;
sbit AC   = 0xD6;
sbit CY   = 0xD7;

/*-----------------------------------------------
Interrupt Vectors:
Interrupt Address = (Number * 8) + 3
-----------------------------------------------*/
#define IE0_VECTOR      0  /* 0x03 */
#define TF0_VECTOR      1  /* 0x0B */
#define IE1_VECTOR      2  /* 0x13 */
#define TF1_VECTOR      3  /* 0x1B */
#define SIO_VECTOR      4  /* 0x23 */
#define PFW_VECTOR      5  /* 0x2B */

/*-----------------------------------------------
-----------------------------------------------*/
#endif
```