Application Note 135
DS1631 Temperature Reading with the DS5000/8051

INTRODUCTION

The DS1631 Digital Thermometer and Thermostat provides 9-bit temperature readings in "counter mode" which indicate the temperature of the device. User-defined temperature settings are stored in nonvolatile memory so parts may be programmed prior to insertion in a system. Temperature settings and readings are communicated via a 2-wire serial interface. It has an alarm output, so the device can also act as a thermostat. The DS1631, which incorporates a 2-wire interface can be controlled using an 8051-compatible DS5000 Secure Microcontroller. The DS1631 is connected directly to the I/O port on the DS5000 microcontroller, and the 2-wire handshaking and temperature readings are handled by low-level software drivers as shown in this document.

READING TEMPERATURE WITH THE DS5000/8051

Data is transmitted over the 2-wire bus, MSB first. Temperature data may be transferred either as a single byte with a resolution of 1°C or as two bytes 0.5°C resolution. The second byte would contain the value of the least significant (0.5°C) bit of the temperature reading. The temperature reading of less than 0°C is obtained by calculating the two's complement of the data. See Table 1.0.

TABLE 1.0

Temperature/Data Relationships

S	2^6	2^5	2^4	2^3	2^2	2^1	2^0	MSB
MSb				(unit = °C)			LSb	
2^{-1}	0	0	0	0	0	0	0	LSB

TEMPERATURE	DIGITAL OUTPUT (Binary)	DIGITAL OUTPUT (Hex)
+125°C	01111101 00000000	7D00h
+25°C	00011001 00000000	1900
0.5°C	00000000 10000000	0080
0°C	00000000 00000000	0000
-0.5°C	11111111 10000000	FF80
-25°C	11100111 00000000	E700h
-55°C	11001001 00000000	C900h

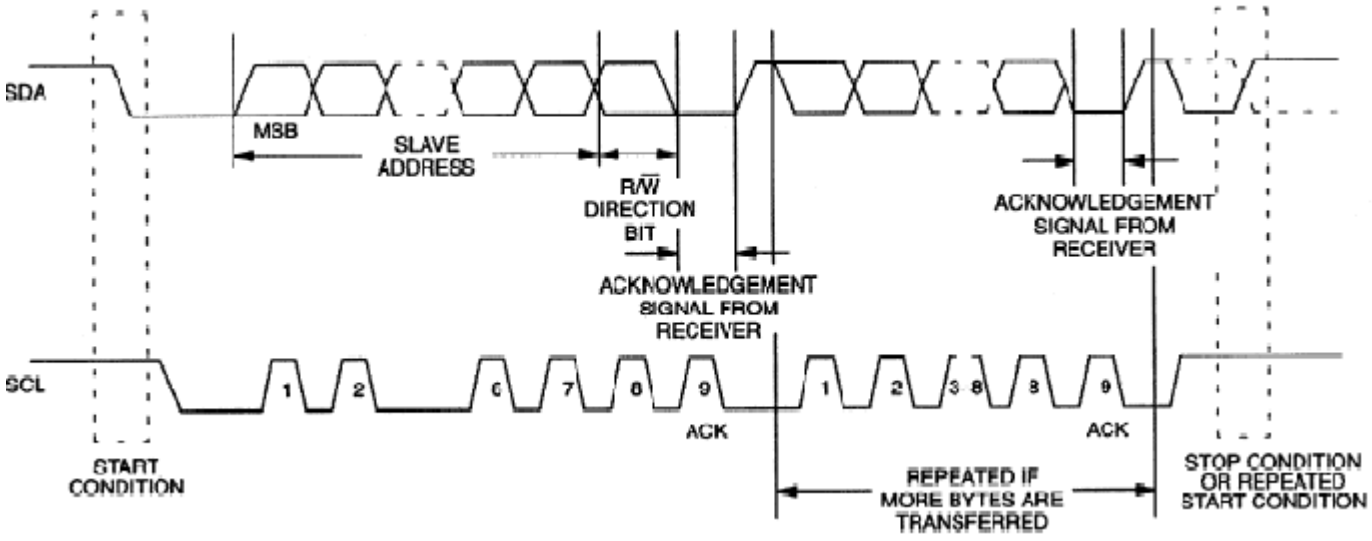
Higher resolutions can be obtained by loading the value of the slope accumulator into the count register. This value may then be read, yielding the number of counts per degree C. The actual temperature can then be calculated using the following formula:

$$\text{TEMPERATURE} = \text{TEMP\_READ} - 0.25 + \frac{(\text{COUNT\_PER\_C} - \text{COUNT\_REMAIN})}{\text{COUNT\_PER\_C}}$$

The DS1631 supports bidirectional 2-wire bus and data transmission protocol. A device that sends data onto the bus is defined as a “transmitter”, and the device receiving the data is known as the “receiver.”

The device that controls the message is called the bus “master.” All devices that are controlled by the bus “master” are known as “slaves” on the 2-wire bus. The bus must be controlled by a master device that generates the serial clock (SCL). Bidirectional data is sent and received via the serial data line (SDA). The bus master also sends the START and STOP conditions. The DS1631 operates as a slave on the 2-wire bus. Connections to the bus are made via the open-drain I/O lines SDA and SCL. See Figure 1.0 for 2-wire serial bus data transfer.

FIGURE 1.0 **DATA TRANSFER ON 2-WIRE SERIAL BUS**



TEMPERATURE CONTROL OF THE DS1631

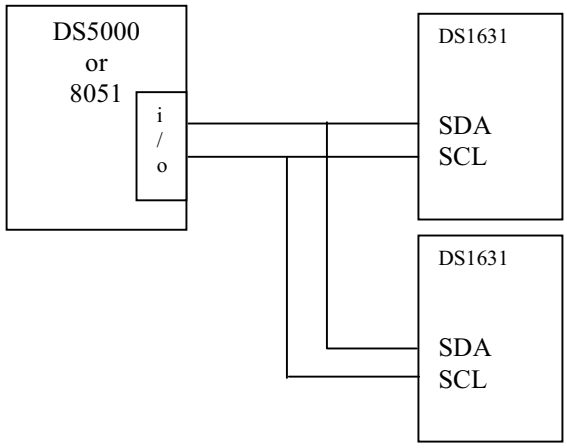
The thermostat output of the DS1631 allows it to directly control heating and cooling devices. T_{OUT} is driven high if the device exceeds a predefined limit set within the TH Register. The output T_{OUT} can be used to indicate that a high temperature tolerance boundary has been met or exceeded. T_{OUT} is driven low when the temperature of the device falls below the limit set in the TL Register. For typical thermostat operation, the DS1631 will operate in continuous mode. However, for applications where only one reading is needed at certain times or to conserve power, the one-shot mode may be used. Note that the thermostat output (T_{OUT}) will remain in the state after the last valid temperature conversion cycle when operating in one-shot mode.

HARDWARE CONFIGURATION

The 2-Wire bus is comprised of two signals. These are SCL (clock) signal and the SDA (address/data) signal. All data transfers are initiated by driving the SDA input high with the clock high. A clock cycle is

a sequence of a falling edge followed by a rising edge. For data inputs, the data must be valid during the rising edge of the clock cycle. Data bits are output on the falling edge of the clock and remain valid through the rising edge. Figure 2.0 illustrates the device connection to the microcontroller programmable input/output port. For illustration purposes, only two devices are attached to the 2-Wire bus. However, three address lines are provided (A0, A1, A2), which allow up to eight devices on a single bus.

FIGURE 2.0 **HARDWARE BLOCK DIAGRAM**



Note: Up to eight devices may share the same 2-wire bus, if each device is set to a different address (0-7).

The actual hardware configuration used to simulate the microcontroller environment is provided in Appendix B, as shown in the program header file. Note that the DS5000/DS2250 is run at a frequency of 11.05949 MHz. The DS232A is used to handle the PC to microcontroller interface. As shown in Appendix B, the 2-Wire connection is made via the I/O port P0. Port I/O P1 or P2 can be used to report status or to power a peripheral reporting device such as an LCD.

SOFTWARE CONTROL

The C source code program “ds1631.c” used to test the hardware and perform the temperature readings is provided in Appendix A. Note that the header file “reg50002w.h” is also provided in Appendix B.

Development software tools can be downloaded from the Dallas Semiconductor website: ftp://ftp.dalsemi.com/pub/microcontroller/

APPENDIX A — C SOURCE MICROCONTROLLER SOFTWARE

```
/*---------------------------------------------------------------------*/
// ds1631.c -- Functions for the Dallas Semiconductor DS1631
// Two-Wire Temperature Sensor
//  Designed for 8051 microcontrollers
// This code was developed using the DS5000/DS2250
//
/*---------------------------------------------------------------------*/
//#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 <ds50002w.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     */
/*---------------------------------------------------------------------*/
/*---------------------------------------------------------------------*/
#ifndef READ
   #define READ   1
#endif

#ifndef WRITE
   #define WRITE  0
#endif

#ifndef I2CCLK
   #define I2CCLK 0xA0
#endif

short int   Config_Data;                /* Config. Reg. Data                        */
unsigned char   Temp_Low;               /* Temp_Low Data                            */
float                   temp_c;                                                      /* temperature in deg. C    */
float                   temp_f;                                                     /* temperature in deg. F     */
unsigned int        count_remain;               /* used for getting .5 deg.  */
unsigned int        count_per_c;                /* used for getting .5 deg.  */
unsigned char   temp_and_half_bit;      /* temp byte and half bit       */
unsigned char   sign_bit;               /* sign bit    */
unsigned char   half_bit;               /* sign bit    */
unsigned char   read_slope;
unsigned char   Select_Type;    /* Function variable                */
short int  Data1;
short int   Data2;                  /* Config. Reg.*/
unsigned char b, k, result;
unsigned char _i2c_error;      // bit array of error types

void I2CSendDataByte1(unsigned char bt);
void I2CSendDataByte2(unsigned char bt);
void I2C_GetTemp02(void);
void I2C_GetTemp01(void);
void          _I2CBitDly(void);
```

```
void        _I2CSCLHigh(void);
void        I2CSendAddr(unsigned char addr, unsigned char rd);
void        I2CSendByte(unsigned char bt);
unsigned char _I2CGetByte(unsigned char lastone);
void        I2CSendStop(void);
#define I2CGetByte()     _I2CGetByte(0)
#define I2CGetLastByte() _I2CGetByte(1)
/*---------------------------------------------------------------------*/
/*                  M   M  AAA   III  N   N                            */
/*                  M   M A   A   I   N   N                            */
/*                  MM MM A   A   I   NN  N                            */
/*                  MM MM AAAAA   I   N N N                            */
/*                  M M M A   A   I   N  NN                            */
/*                  M   M A   A   I   N   N                            */
/*                  M   M A   A  III  N   N                            */
/*---------------------------------------------------------------------*/
/*---------------------------------------------------------------------*/
void main (void)  {                 /* main program                    */

/*---------------------------------------------------------------------*/
/*     Local variables                                                 */
/*---------------------------------------------------------------------*/

Select_Type  = 0; // initialize command selection

/*---------------------------------------------------------------------*/
/*     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 DS1631 Two-Wire Device  banner                          */
/*---------------------------------------------------------------------*/
printf ("\n");
printf ("           Dallas Semiconductor - Systems Extension\n");
printf ("           This program selects between two DS1631 devices on the same bus.\n");
printf ("           Updated  Code May, 2001 \n");
printf ("           [C Program for DS500x or 8051 Compatible Microcontroller]");
printf("\n\n");
printf("\n*****************************************************************\n");
printf ("           Select Menu Option\n");
```

```
printf ("          1. Read Temperature, Device 0\n");
printf ("          2. Read Configuration Register, Device 0\n");
printf ("          3. Write Configuration Register = 00h, Clear Flags, Device 0\n");
printf ("          4. Read TH and TL Registers, Device 0\n");
printf ("          5. Write TH=30.5 degrees, Device 0: Write TL=10 Device 0\n");
printf ("          6. Read Temperature, Device 1\n");
printf ("          7. Read Configuration Register, Device 1\n");
printf ("          8. Write Configuration Register = 00h, Clear Flags, Device 1\n");
printf ("          9. Write TH=40.5 degrees, Device 1: Write TL=0.5 Device 1\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");
printf ("\n\n");

do {
 /*-------------------------------------------------------*/

Select_Type = getchar(); // halt to view data

switch(Select_Type)
{

        case '1':  printf ("\n          1. Read Temperature, Device 00\n");

                                I2C_GetTemp01();
                                  break;

        case '2':  printf ("\n          2. Read Configuration Register, Device 00\n");
                                I2CSendStop(); //send stop
                                I2CSendAddr(0x90,WRITE); // device address
                                I2CSendByte(0xAC);  // command byte access Config. reg.
                                I2CSendAddr(0x90,READ);  // send read command
                                Config_Data = I2CGetLastByte();  // get last byte
                                printf("\n\nconfig data = %02X\n", Config_Data);  //Print Config. Data
                                I2CSendStop(); //send stop
                                break;

        case '3':  printf ("\n  3. Write Configuration Register = 00h, Clear Flags, Device 00\n");
                                I2CSendAddr(0x90,WRITE); // device address
                                I2CSendDataByte1(0xAC);  // command byte access Config. reg.
                                I2CSendDataByte2(0x00);  // data send 00
                                 _I2CBitDly(); // wait
                                I2CSendAddr(0x90,WRITE); // device address
                                I2CSendByte(0xAC);  // command byte access Config. reg.
                                I2CSendAddr(0x90,READ); // send read command
                                Config_Data = I2CGetLastByte();  //get last byte
                                printf("\n\nConfig Data = %02X\n", Config_Data);  //Print Config. Data
                                I2CSendStop(); //send stop
                                break;

        case '4':  printf ("  4. Read TH and TL Registers: Device 0, Device 1\n");
                                Data1 = 0;
                                Data2=0;
                                I2CSendAddr(0x90,WRITE); // command byte  device address
                                I2CSendByte(0xA1);  // command byte  read temp
                                I2CSendAddr(0x90,READ); // command byte read temp
                                Data1 = I2CGetByte();      /* read 1st byte                */
                                Data2 = I2CGetLastByte();  //last byte
                                printf( "\nTemp. High Data Device 0 : MSB: %02X :LSB: %02X\n", Data1,Data2);
```

```
                        I2CSendStop();
                        Data1 = 0;
                        Data2=0;
                        I2CSendAddr(0x90,WRITE); // command byte  device address
                        I2CSendByte(0xA2);  // command byte  read temp
                        I2CSendAddr(0x90,READ); // command byte read temp
                        Data1 = I2CGetByte();      /* read 1st byte                */
                        Data2 = I2CGetLastByte();  // last byte
                        printf( "\nTemp. Low Data Device 0: MSB: %02X  :LSB: %02X  \n", Data1,Data2);

                        I2CSendStop();
                        _I2CBitDly(); // wait
                        I2CSendAddr(0x92,WRITE); // command byte  device address
                        I2CSendByte(0xA1);  // command byte  read temp high register
                        I2CSendAddr(0x92,READ); // command byte read temp
                        Data1 = 0;
                        Data2 = 0;
                        Data1 = I2CGetByte();      /* read 1st byte                */
                        Data2 = I2CGetLastByte();  //last byte
                        printf( "\nTemp. High Data Device 1: MSB: %02X :LSB: %02X\n", Data1,Data2);
                        I2CSendStop();
                        _I2CBitDly(); // wait
                        I2CSendAddr(0x92,WRITE); // command byte  device address
                        I2CSendByte(0xA2);  // command byte  read temp LOW register
                        I2CSendAddr(0x92,READ); // command byte read temp
                        Data1 = 0;
                        Data2 = 0;
                        Data1 = I2CGetByte();      /* read 1st byte                */
                        Data2 = I2CGetLastByte();  //last byte
                        printf( "\nTemp. Low Data Device 1: MSB: %02X :LSB: %02X\n", Data1,Data2);

                        I2CSendStop();
                        break;

case '5':   printf (" 5. Write TH=30.5 degrees, Device 0: Write TL=10 Device 0\n");
                        I2CSendAddr(0x90,WRITE); // command byte  device address
                        I2CSendDataByte1(0xA1);    /* Write to Temp. High Register */
                        I2CSendDataByte1(0x1E);    /* set to 1Eh     */
                        I2CSendDataByte2(0x80);    /* set to 80h     */
                        I2CSendStop();  //end transmission
                        _I2CBitDly(); // wait
                        I2CSendAddr(0x90,WRITE); // command byte  device address
                        I2CSendByte(0xA1);  // command byte  read temp high register
                        I2CSendAddr(0x90,READ); // command byte read temp
                        Data1 = 0;
                        Data2 = 0;
                        Data1 = I2CGetByte();      /* read 1st byte                */
                        Data2 = I2CGetLastByte();  //last byte
                        printf( "\nTemp. High Data : %02X %02X\n", Data1,Data2);
                        I2CSendStop();
                        I2CSendAddr(0x90,WRITE); // command byte  device address
                        I2CSendDataByte1(0xA2);    /* Write to Temp. LOW Register */
                        I2CSendDataByte1(0x10);    /* set to 10h     */
                        I2CSendDataByte2(0x00);    /* set to 00h     */
                        I2CSendStop();  //end transmission
                        _I2CBitDly(); // wait
                        I2CSendAddr(0x90,WRITE); // command byte  device address
                        I2CSendByte(0xA2);  // command byte  read temp LOW register
                        I2CSendAddr(0x90,READ); // command byte read temp
                        Data1 = 0;
```

```
                        Data2 = 0;
                        Data1 = I2CGetByte();      /* read 1st byte                */
                        Data2 = I2CGetLastByte();  //last byte
                        printf( "\nTemp. LOW Data : %02X %02X\n", Data1,Data2);
                        I2CSendStop();
                        break;


case '6':   printf ("    6. Read Temperature, Device 01\n");

                        I2C_GetTemp02();
                          break;

case '7':  printf ("\n  7. Read Configuration Register, Device 01\n");
                        I2CSendStop();//send stop
                        I2CSendAddr(0x92,WRITE);// device addresss
                        I2CSendByte(0xAC);  // command byte access Config. reg.
                        I2CSendAddr(0x92,READ);  // send read command
                        Config_Data = I2CGetLastByte();  //get last byte
                        printf("\n\nConfig Data = %02X\n", Config_Data);  //Print Config. Data
                        I2CSendStop(); //send stop
                        break;

case '8':  printf ("\n 8. Write Configuration Register = 00h, Clear Flags, Device 01\n");
           I2CSendAddr(0x92,WRITE); // device address
                        I2CSendDataByte1(0xAC);  // command byte access Config. reg.
                        I2CSendDataByte2(0x00);  // send command continuous conversions
                        I2CSendStop(); //send stop
                         _I2CBitDly(); // wait
                        I2CSendStop(); //send stop
                        I2CSendAddr(0x92,WRITE);  // device address
                        I2CSendByte(0xAC);  // command byte access Config. reg.
                        I2CSendAddr(0x92,READ);  //send read command
                        Config_Data = I2CGetLastByte();  // get the last byte
                        printf("\n\nConfig. Data = %02X\n", Config_Data);  //Print Config. Data
                        I2CSendStop();  //send stop
                        break;

case '9':   printf (" 9. Write TH=40.5 degrees, Device 1: Write TL=0.5 Device 1\n");
                        I2CSendAddr(0x92,WRITE); // command byte  device address
                        I2CSendDataByte1(0xA1);    /* Write to Temp. High Register */
                        I2CSendDataByte1(0x28);    /* set to 28h     */
                        I2CSendDataByte2(0x80);    /* set to 80h     */
                        I2CSendStop();  //end transmission
                        _I2CBitDly(); // wait
                        I2CSendAddr(0x92,WRITE); // command byte  device address
                        I2CSendByte(0xA1);  // command byte  read temp high register
                        I2CSendAddr(0x92,READ); // command byte read temp
                        Data1 = 0;
                        Data2 = 0;
                        Data1 = I2CGetByte();      /* read 1st byte                */
                        Data2 = I2CGetLastByte();  //last byte
                        printf( "\nTemp. High Data : %02X %02X\n", Data1,Data2);
                        I2CSendStop();
                        I2CSendAddr(0x92,WRITE); // command byte  device address
                        I2CSendDataByte1(0xA2);    /* Write to Temp. LOW Register */
                        I2CSendDataByte1(0x00);    /* set to 00h     */
                        I2CSendDataByte2(0x80);    /* set to 80h     */
                        I2CSendStop();  //end transmission
                        _I2CBitDly(); // wait
```

```
                                        I2CSendAddr(0x92,WRITE); // command byte  device address
                                        I2CSendByte(0xA2);  // command byte  read temp LOW register
                                        I2CSendAddr(0x92,READ); // command byte read temp
                                        Data1 = 0;
                                        Data2 = 0;
                                        Data1 = I2CGetByte();      /* read 1st byte                */
                                        Data2 = I2CGetLastByte();  //last byte
                                        printf( "\nTemp. Low Data : %02X %02X\n", Data1,Data2);
                                        I2CSendStop();
                                        break;

                default:  printf ("\n           Typo: Select Another Menu Option\n");
                                break;
};              /* end switch*/

}while(1); //keep looping

}//    End Main Program

void _I2CBitDly(void)         // wait 4.7uS, or thereabouts =5
{                             // tune to xtal. This works at 11.0592MHz
            unsigned int time_end = 10;
            unsigned int index;
            for (index = 0; index < time_end; index++);
 return;
}


void _I2CSCLHigh(void)        // Set SCL high, and wait for it to go high
{
 register int err;
 SCL = 1;
 while (! SCL)
 {
  err++;
  if (!err)
  {
   _i2c_error &= 0x02;     // SCL stuck, something's holding it down
   return;
  }
 }
}

void I2CSendAddr(unsigned char addr, unsigned char rd)
{
            SCL = 1;
 _I2CBitDly();
 SDA = 0;              // generate start
 _I2CBitDly();
 SCL = 0;
 _I2CBitDly();
 I2CSendByte(addr+rd);  // send address byte

}

void I2CSendByte(unsigned char bt)
{

 register unsigned char i;
  for (i=0; i<8; i++)
```

```
  {
  if (bt & 0x80) SDA = 1;      // Send each bit, MSB first changed 0x80 to 0x01
   else SDA = 0;
   _I2CSCLHigh();
    _I2CBitDly();
    SCL = 0;
    _I2CBitDly();
    bt = bt << 1;

  }
  SDA = 1;                    // Check for ACK
  _I2CBitDly();
  _I2CSCLHigh();
  _I2CBitDly();
  if (SDA)
     _i2c_error &= 0x01;          // Ack didn't happen, may be nothing out there
//printf("\n_ic2_error %x\n", _i2c_error); // For debug purposes.
  SCL = 0;
  _I2CBitDly();
  SDA = 1;  // end transmission
  SCL = 1;
}
void I2CSendDataByte1(unsigned char bt)
{

  int i;
    for (i=0; i<8; i++)
  {
  if (bt & 0x80) SDA = 1;      // Send each bit, MSB first changed 0x80 to 0x01
   else SDA = 0;
   _I2CSCLHigh();
    _I2CBitDly();
    SCL = 0;
    _I2CBitDly();
    bt = bt << 1;
  }
  //SDA = 1;                    // Check for ACK
  _I2CBitDly();
  _I2CSCLHigh();
  _I2CBitDly();
  if (SDA)
     _i2c_error &= 0x01;          // Ack didn't happen, may be nothing out there
//printf("\n_ic2_error %x\n", _i2c_error);
  SCL = 0;
  _I2CBitDly();
 // SDA = 1;  // end transmission
  //SCL = 1;
}
void I2CSendDataByte2(unsigned char bt)
{

 int i;
    for (i=0; i<8; i++)
  {
  if (bt & 0x80) SDA = 1;      // Send each bit, MSB first changed 0x80 to 0x01
   else SDA = 0;
   _I2CSCLHigh();
    _I2CBitDly();
    SCL = 0;
    _I2CBitDly();
```

```
    bt = bt << 1;

  }
  SDA = 1;                    // listen for ACK
  _I2CBitDly();
  _I2CSCLHigh();
  _I2CBitDly();
  if (SDA)
     _i2c_error &= 0x01;          // Ack didn't happen, may be nothing out there
//printf("\n_ic2_error %x\n", _i2c_error);
  SCL = 0;
  _I2CBitDly();
 // SDA = 1;  // end transmission
  SCL = 1;
}
unsigned char _I2CGetByte(unsigned char lastone) // lastone == 1 for last byte
{
 register unsigned char i, res;
  res = 0;
  for (i=0;i<8;i++)          // Each bit at a time, MSB first
  {
    _I2CSCLHigh();
    _I2CBitDly();
    res *= 2;
    if (SDA) res++;
    SCL = 0;
    _I2CBitDly();
  }
  SDA = lastone;             // Send ACK according to 'lastone'
  _I2CSCLHigh();
  _I2CBitDly();
  SCL = 0;
  SDA = 1;   // end transmission
  SCL=1;
  _I2CBitDly();
  return(res);
}

void I2CSendStop(void)
{
  SDA = 0;
  _I2CBitDly();
  SCL = 1;
  _I2CBitDly();
  SDA = 1;
  _I2CBitDly();
}



void I2C_GetTemp01(void)
{

              I2CSendAddr(0x90,WRITE); // address 000  device 1
              I2CSendByte(0xEE);  // command byte start conversion
              I2CSendStop();  //send stop
             _I2CBitDly(); // wait
            I2CSendAddr(0x90,WRITE); // command byte  device address
I2CSendByte(0xAA);  // command byte  read temp
I2CSendAddr(0x90,READ); // command byte read temp
```

```
temp_and_half_bit = I2CGetByte(); // byte one
half_bit = I2CGetLastByte();  //byte two
/*  Modify Temp Data  */

sign_bit=            temp_and_half_bit & 0x80;
if ( sign_bit != 0 )            printf ( "\nSIGN BIT IS SET... SEE=%02X \n", sign_bit );
if ( sign_bit != 0 )temp_and_half_bit=(~temp_and_half_bit) + 1;/* twos complement */
if ( sign_bit != 0 )            printf ( "\nTwos complement ==%02X \n", temp_and_half_bit  );
if ( sign_bit != 0 )half_bit=(~half_bit) + 1;  /* twos complement */
if ( sign_bit != 0 )            printf ( "\nTwos complement half bit ==%02X \n", half_bit  );
if ( sign_bit != 0 )  sign_bit = -1;
/*-------------------------------------------------------------------------*/
/*      Get count remain & count per C for .5 resolution                   */
/*-------------------------------------------------------------------------*/
I2CSendAddr(0x90,WRITE); // command byte read counter
I2CSendByte(0xA8);  // command byte  counter
I2CSendAddr(0x90,READ);                         /* read count remain          */
count_remain = I2CGetLastByte();

if ( sign_bit != 0 )count_remain=(~count_remain) + 1;  /* twos complement */
if ( sign_bit != 0 )            printf ( "\nTwos complement half bit ==%02X \n",count_remain );
I2CSendAddr(0x90,WRITE); // command byte read slope
I2CSendByte(0xA9);  // command byte  slope
I2CSendAddr(0x90,READ);                         /* read slope        */
read_slope = I2CGetLastByte();

I2CSendAddr(0x90,WRITE); // command byte read counter
I2CSendByte(0xA8);  // command byte  counter
I2CSendAddr(0x90,WRITE); // command byte read scounter
I2CSendByte(read_slope);  // command byte put slope into count remain

I2CSendAddr(0x90,WRITE); // command byte read counter

I2CSendByte(0xA8);  // command byte  counter
I2CSendAddr(0x90,READ); // command byte read counter
count_per_c = I2CGetLastByte();    /* read 2nd byte                */

/*-------------------------------------------------------------------------*/
/*      Calculate øC and øF                                                */
/*-------------------------------------------------------------------------*/
            if ( count_per_c == 0 ) count_per_c = 1;
            if(count_remain > count_per_c) count_remain = 1;
            temp_c = (float)temp_and_half_bit-0.25 + (count_per_c-count_remain)/(float) count_per_c;
            if ( sign_bit != 0 )temp_c = -((float)temp_and_half_bit-0.25
            + (count_per_c-count_remain)/(float) count_per_c);
            temp_f = temp_c * 9/5 + 32;

/*-------------------------------------------------------------------------*/
/*      Display temp to CRT                                                */
/*-------------------------------------------------------------------------*/

printf( "\nTempC=%5.1f \n", temp_c );  // print temp. C
printf( "\nTempF=%5.1f \n", temp_f );  // print temp. F



}
void I2C_GetTemp02(void)
{
```

```
            I2CSendAddr(0x92,WRITE); // address 000  device 1
            I2CSendByte(0xEE);  // command byte start conversion
            I2CSendStop();  //send stop
           _I2CBitDly(); // wait
           I2CSendAddr(0x92,WRITE); // command byte  device address
I2CSendByte(0xAA);  // command byte  read temp
I2CSendAddr(0x92,READ); // command byte read temp
temp_and_half_bit = I2CGetByte(); // byte one
half_bit = I2CGetLastByte();  //byte two
/*  Modify Temp Data  */

sign_bit=           temp_and_half_bit & 0x80;
if ( sign_bit != 0 )            printf ( "\nSIGN BIT IS SET... SEE=%02X \n", sign_bit );
if ( sign_bit != 0 )temp_and_half_bit=(~temp_and_half_bit) + 1;/* twos complement */
if ( sign_bit != 0 )            printf ( "\nTwos complement ==%02X \n", temp_and_half_bit  );
if ( sign_bit != 0 )half_bit=(~half_bit) + 1;  /* twos complement */
if ( sign_bit != 0 )            printf ( "\nTwos complement half bit ==%02X \n", half_bit  );
if ( sign_bit != 0 )  sign_bit = -1;
/*-------------------------------------------------------------------------*/
/*      Get count remain & count per C for .5 resolution                   */
/*-------------------------------------------------------------------------*/
I2CSendAddr(0x92,WRITE); // command byte read counter
I2CSendByte(0xA8);  // command byte  counter
I2CSendAddr(0x92,READ);                         /* read count remain       */
count_remain = I2CGetLastByte();

if ( sign_bit != 0 )count_remain=(~count_remain) + 1;  /* twos complement */
if ( sign_bit != 0 )            printf ( "\nTwos complement half bit ==%02X \n",count_remain );
I2CSendAddr(0x92,WRITE); // command byte read slope
I2CSendByte(0xA9);  // command byte  slope
I2CSendAddr(0x92,READ);                         /* read slope      */
read_slope = I2CGetLastByte();

I2CSendAddr(0x92,WRITE); // command byte read counter
I2CSendByte(0xA8);  // command byte  counter
I2CSendAddr(0x92,WRITE); // command byte read scounter
I2CSendByte(read_slope);  // command byte put slope into count remain

I2CSendAddr(0x92,WRITE); // command byte read counter

I2CSendByte(0xA8);  // command byte  counter
I2CSendAddr(0x92,READ); // command byte read counter
count_per_c = I2CGetLastByte();    /* read 2nd byte                */

/*-------------------------------------------------------------------------*/
/*      Calculate øC and øF                                                */
/*-------------------------------------------------------------------------*/
            if ( count_per_c == 0 ) count_per_c = 1;
            if(count_remain > count_per_c) count_remain = 1;
            temp_c = (float)temp_and_half_bit-0.25 + (count_per_c-count_remain)/(float) count_per_c;
            if ( sign_bit != 0 )temp_c = -((float)temp_and_half_bit-0.25
            + (count_per_c-count_remain)/(float) count_per_c);
            temp_f = temp_c * 9/5 + 32;

/*-------------------------------------------------------------------------*/
/*      Display temp to CRT                                                */
/*-------------------------------------------------------------------------*/

printf( "\nTempC=%5.1f \n", temp_c );  // print temp. C
```

```
printf( "\nTempF=%5.1f \n", temp_f );  // print temp. F



}
//  Note: This program represents one possibility for accessing device registers
//   the user should feel free to explore other, more efficient means and not except
//   this example as the single and final approach. This software provides no warranties.
```

APPENDIX B — HEADER FILE SOURCE CODE

```
/*---------------------------------------------------------------------------
DS50002w.H

Header file for Dallas Semiconductor DS5000/8051.
---------------------------------------------------------------------------*/

#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_1 = 0x80;
//sbit P0_2 = 0x81;
sbit SDA = 0x80;
sbit SCL = 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
```