Keywords: 2-wire, 2wire, temperature sensors, digital temperature sensors, temperature sensor IC, microcontrollers, micro-controller, DS1631

APPLICATION NOTE 135

Interfacing to the DS1631 Digital Thermometer and Thermostat in a Microcontroller Environment

Feb 22, 2002

Abstract: This application note introduces the user to software for interfacing a DS5000 (8051 compatible) microcontroller to the DS1631 temperature sensor. The DS1631 incorporates a standard 2-wire serial digital interface. Software code is provided that can be used to provide all types of functional access to the DS1631 including reading the temperature register, writing the thermostat thresholds, and setting the device configuration.

Introduction

The DS1631 is a digital thermometer that provides 9, 10, 11, or 12-bit temperature readings over a -55°C to +125°C range, and has ±0.5°C accuracy from 0°C to +70°C with 3.0V ≤ V_{DD} ≤ 5.5V. The DS1631 also provides thermostatic functionality with user-defined trip points (T_H and T_L). Three address pins allow up to eight DS1631s to function on the same bus.

Communication with the DS1631 is achieved via a 2-wire serial interface. This application note presents 'C' source code that allows a PC to communicate with the DS1631 via an 8051-compatible DS5000 microcontroller. Detailed specifications and operating instructions for the DS1631 can be found in the datasheet.

Hardware Configuration

The DS1631 SDA (serial data) and SCL (serial clock) pins can be connected directly to the I/O port on the DS5000 microcontroller. The DS1631 SDA pin is an open drain I/O, so the SDA line must be pulled high by a pullup resistor. Since the DS5000 microcontroller clock output is also open-drain, a pull up resistor is also required on the SCL line. **Figure 1** shows a circuit diagram with two DS1631s (addresses 1001000 and 1001001) connected to the bus.

The DS5000 configuration is provided in the header file in Appendix B. Note that the DS5000 is run at a frequency of 11.05949MHz. A 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 of the DS5000. I/O port P1 or P2 can be used to report status or to power a peripheral reporting device such as an LCD.

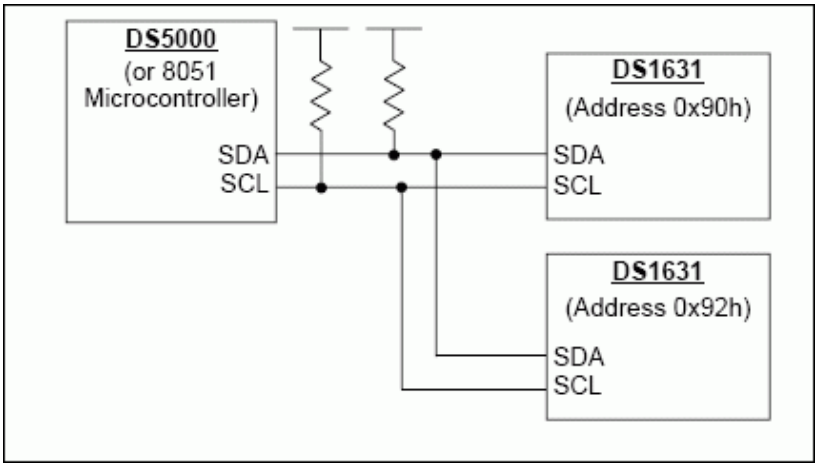

Figure 1. Circuit diagram for two DS1631 devices on the same two-wire bus.

Software Communication with the DS1631

The following sections contain DS1631 'C' source code examples. Appendix A provides a complete C program listing for testing the hardware and communicating with the DS1631.

START and STOP Condition

START and STOP conditions are used by the microcontroller to signal the beginning and end, respectively, of a 2-wire communication sequence. To produce a START condition, the SDA line is pulled from high to low while SCL is high, and for a STOP condition SDA transitions low to high while SCL is high. **Figure 2a** shows sample 'C' code for generating a START condition and **Figure 2b** shows sample code for generating a STOP condition.

```
void I2CSendStart(void)
{
        SCL = 1;        // Take SCL high
        _I2CBitDly();   // Wait
        SDA = 0;        // Pull SDA low
        _I2CBitDly();   // Wait
        SCL = 0;        // Take SCL low
        _I2CBitDly();   // Wait
}
```

Figure 2a. START Example.

```
void I2CSendStop(void)
{
        SDA = 0;        // Pull SDA low
        _I2CBitDly();   // Wait
        SCL = 1;        // Take SCL high
        _I2CBitDly();   // Wait
        SDA = 1;        // Release SDA to pullup
        _I2CBitDly();   // Wait
}
```

Figure 2b. STOP Example.

Writing to the DS1631

The master has write access to the 1-byte configuration register and the 2-byte T_H and T_L registers. Therefore, when writing to the configuration register, the master must send one byte of data, and when writing to the T_H or T_L registers the master must send two bytes of data. **Figure 3** shows example 'C' code for writing to the configuration register. The SendAddr routine causes a START condition to be generated followed by a control byte that contains the DS1631 address and has the Read/Write bit set for "write". Next the SendByte routine is used to issue an Access Config command (ACh). This is followed by the data byte being written to the configuration register. The sequence is completed with a STOP condition.

```
void WriteConfig(unsigned char Address, unsigned char Data)          //Pass  Device Address such as 0x90h or 0x92h
{
                                        // Routine to write data to Config Register
        I2CSendAddr(Address,WRITE);             // send START and control byte
        I2CSendByte (0xAC);                     // send Access Config command byte
        I2CSendByte (Data);                     // send data
        I2CSendStop();                          // send STOP
}
```

Figure 3. Code example for writing to the DS1631.

Reading from the DS1631

The master can read data from the 1-byte configuration register and the 2-byte temperature, T_H and T_L registers. **Figure 4** shows example 'C' code for reading the configuration register. The SendAddr routine generates a START followed by a control byte with the Read/Write bit set for "write". Next the Access Config command (ACh) is sent. This is followed by another START plus a control byte, but this time the Read/Write bit is set to "read". The I2CGetByte(1) routine reads and saves the 1-byte configuration register. The "1" that is passed to the routine indicates that this byte is the last (and in the case the only) byte being read. This is necessary so that a NACK instead of an ACK is sent to the DS1631 after the byte is received. For 2-byte reads, a "0" is passed to the I2CGetByte subroutine after the first byte and a "1" is sent after the second byte. The sequence is completed with a STOP condition.

```
void GetConfig(unsigned char Address)              //Pass an Address such as 0x90h or 0x92h
{
                                                   // Routine to read data from Config Register
        I2CSendAddr(Address,WRITE);                // send START and control byte
        I2CSendByte(0xAC);                         // send Access Config command byte
        I2CSendAddr(Address,READ);                 // send repeat START and control byte
        Config = I2CGetByte(1);                    // read Config Register
        I2CSendStop();                             // send STOP
}
```

Figure 4. Code example for reading from the DS1631.

Calculating the Temperature

After each temperature conversion, the DS1631 stores the digital temperature as a 16-bit two's complement number in the 2-byte temperature register. **Figure 5** shows example 'C' code for initiating a temperature conversion and then reading the temperature register and calculating decimal Centigrade and Fahrenheit values from the digital value. To initiate a temperature conversion, a START is sent followed by a control byte with the Read/Write bit set for "write". Next a Start Convert T command (51h) is sent followed by a STOP condition. To read the temperature register, a START is sent followed by a control byte with the Read/Write bit set for "write". Next the Read Temperature command (AAh) is sent followed by another START plus a control byte with the Read/Write bit is set to "read". I2CGetByte(0) reads the temperature MSB and sends an ACK to the DS1631. I2CGetByte(1) then reads the temperature LSB and sends a NACK to the DS1631. The communication sequence is completed with a STOP condition. The final code segment converts the 2-byte digital reading to decimal values.

```
void GetTemp(unsigned char Address)                // Pass an Address such as 0x90h or 0x92h
{
                                                   // Initiate temperature conversion
        I2CSendAddr(Address,WRITE);                // send START and control byte
        I2CSendByte(0x51);                         // send Start Convert Tcommand byte
        I2CSendStop();                             // send STOP
        I2CBitDly();                               // wait
                                                   // Read temperature register
        I2CSendAddr(Address,WRITE);                // send START and control byte
        I2CSendByte(0xAA);                         // send Read Temperature command byte
        I2CSendAddr(Address,READ);                 // send repeat START and control byte
        MSB = I2CGetByte(0);                       // read Temp MSB
        LSB = I2CGetByte(1);                       // read Temp LSB
        I2CSendStop();                             // send STOP
                                                   // Calculate temperature
        if(MSB>=0x80)                              //if sign bit is set, then temp is negative
                temp_c = (float)((MSB<<8 + LSB) - 65536) * 0.0625;
        else
                temp_c = (float)(MSB<<8 + LSB) * 0.0625;

        temp_f =( temp_c * 9/5) + 32;
}
```

Figure 5. Code example for reading from the DS1631.

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
//
//-------------------------------------------------------------------------
// 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 8051 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
//global variables
unsigned char Config; // Config. Reg. Data
float temp_c; // temperature in deg. C
float temp_f; // temperature in deg. F
float TH; // TH byte
float TL; // TL byte
unsigned char MSB; // temp byte MSB
unsigned char LSB; // temp byte LSB
unsigned char Select_Type; // Function variable
//Function Prototypes
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 I2CSendStart(void);
void I2CSendStop(void);
void GetTemp(unsigned char Address);
void GetConfig(unsigned char Address);
void WriteConfig(unsigned char Address, unsigned char Data);
void ReadTHandTL(unsigned char Address);
void WriteTHandTL(unsigned char Address, float TH, float TL);
//-------------------------------------------------------------------------
// MAIN
//-------------------------------------------------------------------------
//-------------------------------------------------------------------------
void main (void)
{
Select_Type = 0; // initialize command selection
//-------------------------------------------------------------------------
// 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 - Battery Management / Thermal \n");
printf (" This program selects between two DS1631 devices on the same bus\n");
printf (" with the addresses of 0x90h and 0x92h.\n");
printf (" Updated Code October 2002 \n");
printf (" [C Program for DS500x or 8051 Compatible Microcontroller]");
printf("\n\n");
printf("\n*********************************************************************\n");
printf (" Select Menu Option\n");
printf (" 0. Read Temperature, Device Address 0x90\n");
printf (" 1. Read Configuration Register, Device Address 0x90\n");
printf (" 2. Write Configuration Register = 00h, Clear Flags, Device Address
0x90\n");
printf (" 3. Read TH and TL Registers, Device Address 0x90\n");
printf (" 4. Write TH=30.5 degrees and Write TL=10, Device Address 0x90\n");
printf (" 5. Read Temperature, Device Address 0x92\n");
printf (" 6. Read Configuration Register, Device Address 0x92\n");
printf (" 7. Write Configuration Register = 00h, Clear Flags, Device Address
0x92\n");
printf (" 8. Read TH and TL Registers, Device Address 0x92\n");
printf (" 9. Write TH=40.5 degrees, Write TL=0.5, Device Address 0x92\n");
printf ("\n\n");
do
{
Select_Type = getchar(); // wait for selection
switch(Select_Type)
{
case '0': printf ("\n 1. Read Temperature, Device Address 0x90\n");
GetTemp(0x90);
break;
case '1': printf ("\n 1. Read Config Register, Device Address 0x90\n");
GetConfig(0x90);
break;
case '2': printf ("\n 2. Write Config Register = 00h, Clear Flags, Device Address
0x90\n");
WriteConfig(0x90, 0x00);
break;
case '3': printf ("\n 3. Read TH and TL Registers, Device Address 0x90\n");
ReadTHandTL(0x90);
break;
case '4': printf (" 4. Write TH=30.5 degrees, Device 0: Write TL=10 Device Address
0x90\n");
WriteTHandTL(0x90, 30.5, 10);
break;
case '5': printf ("\n 5. Read Temperature, Device Address 0x92\n");
GetTemp(0x92);
break;
case '6': printf ("\n 6. Read Configuration Register, Device Address 0x92\n");
GetConfig(0x92);
break;
case '7': printf ("\n 7. Write Configuration Register = 00h, Clear Flags, Device
Address 0x92\n");
WriteConfig(0x92,0x00);
break;
case '8': printf ("\n 8. Read TH and TL Registers: Device Address 0x92\n");
ReadTHandTL(0x92);
break;
case '9': printf ("\n 9. Write TH=40.5 degrees, Device 1: Write TL=0.5 Device
Address 0x92\n");
WriteTHandTL(0x92, 40.5, .5);
break;
default: printf ("\n Select Another Menu Option\n");
```

```
break;
}; // end switch
}while(1); //keep looping
}// End Main Program
void GetTemp(unsigned char Address) //Pass Address 0x90h or 0x92h
{
I2CSendAddr(Address,WRITE); // control byte
I2CSendByte(0x51); // command byte start conversion
I2CSendStop(); // send stop
I2CBitDly(); // wait
I2CSendAddr(Address,WRITE); // control byte
I2CSendByte(0xAA); // command byte read temp
I2CSendAddr(Address,READ); // restart control byte and device address
MSB = I2CGetByte(0); // Temp MSB
LSB = I2CGetByte(1); // Temp LSB
I2CSendStop(); // send stop
// Calculate Temp
if(MSB>=0x80) //if sign bit is set, then temp is negative
temp_c = (float)((MSB<<8 + LSB) - 65536) * 0.0625;
else
temp_c = (float)(MSB<<8 + LSB) * 0.0625;
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 GetConfig(unsigned char Address) //Pass Address 0x90h or 0x92h
{
I2CSendAddr(Address,WRITE); // control byte
I2CSendByte(0xAC); // command byte to access config
I2CSendAddr(Address,READ); // restart
Config = I2CGetByte(1); // Configuration Register
I2CSendStop(); // send stop
//---------------------------------------------------------------------------
// Display Config to CRT
//---------------------------------------------------------------------------
printf( "\nConfig=%02X \n", Config ); // print Config Register Value
}
void WriteConfig(unsigned char Address, unsigned char Data) //Pass Address 0x90h
or 0x92h
{
// Write Data to Config Register
I2CSendAddr(Address,WRITE); // control byte
I2CSendByte (0xAC); // command byte access Config reg.
I2CSendByte (Data); // data to send
I2CSendStop(); // send stop
I2CBitDly(); // wait
GetConfig(Address); // Read Config to verify write
}
void ReadTHandTL(unsigned char Address) //Pass Address 0x90h or 0x92h
{
//Read TH
I2CSendAddr(Address,WRITE); // control byte
I2CSendByte(0xA1); // command byte Access TH
I2CSendAddr(Address,READ); // control byte read temp
MSB = I2CGetByte(0); // read TH MSB
LSB = I2CGetByte(1); // read TH LSB
I2CSendStop();
// Calculate TH
if(MSB>=0x80) //if sign bit is set, then temp is negative
TH = (float)((MSB<<8 + LSB) - 65536) * 0.0625;
else
TH = (float)(MSB<<8 + LSB) * 0.0625;
//Read TL
I2CSendAddr(Address,WRITE); // control byte
I2CSendByte(0xA2); // command byte Access TL
I2CSendAddr(Address,READ); // control byte read temp
MSB = I2CGetByte(0); // read TL MSB
LSB = I2CGetByte(1); // read TL LSB
I2CSendStop();
// Calculate TL
```

```
if(MSB>=0x80) //if sign bit is set, then temp is negative
TL = (float)((MSB<<8 + LSB) - 65536) * 0.0625;
else
TL = (float)(MSB<<8 + LSB) * 0.0625;
//---------------------------------------------------------------------
// Display temp to CRT
//---------------------------------------------------------------------
printf( "\nTH=%5.1f \n", TH ); // print TH
printf( "\nTL=%5.1f \n", TL ); // print TL
}
void WriteTHandTL(unsigned char Address, float TH, float TL)
{
//Write TH
MSB = ((unsigned char)TH<<8) & 0xFF00; //Get MSB of TH (This assumes TH is >0)
LSB = ((unsigned char)TH<<8) & 0xFF; //Get LSB of TH
I2CSendAddr(Address,WRITE); // control byte
I2CSendByte (0xA1); // Access TH
I2CSendByte (MSB); // send MSB
I2CSendByte (LSB); // send LSB
I2CSendStop(); //end transmission
//Write TL
MSB = ((unsigned char)TL<<8) & 0xFF00; //Get MSB of TL(This assumes TH is >0)
LSB = ((unsigned char)TL<<8) & 0xFF; //Get LSB of TL
I2CSendAddr(Address,WRITE); // control byte
I2CSendByte (0xA2); // Access TL
I2CSendByte (MSB); // send MSB
I2CSendByte (LSB); // send LSB
I2CSendStop(); //end transmission
ReadTHandTL(Address);
}
void I2CBitDly(void) // wait approximately 4.7uS
{ // 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)
{
return;
}
}
}
void I2CSendAddr(unsigned char addr, unsigned char rd)
{
I2CSendStart();
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)
```

```
SCL = 0;
I2CBitDly();
SDA = 1; // end transmission
SCL = 1;
}
unsigned char I2CGetByte(unsigned char lastone) // last one == 1 for last byte; 0
for any other 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 I2CSendStart(void)
{
SCL = 1;
I2CBitDly();
SDA = 0;
I2CBitDly();
SCL = 0;
I2CBitDly();
}
void I2CSendStop(void)
{
SDA = 0;
I2CBitDly();
SCL = 1;
I2CBitDly();
SDA = 1;
I2CBitDly();
}
```

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 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
```

Related Parts		
DS1631	High-Precision Digital Thermometer and Thermostat	Free Samples
DS5000	Soft Microcontroller Module	Free Samples

More Information
For Technical Support: http://www.maximintegrated.com/support
For Samples: http://www.maximintegrated.com/samples
Other Questions and Comments: http://www.maximintegrated.com/contact

Application Note 135: http://www.maximintegrated.com/an135
APPLICATION NOTE 135, AN135, AN 135, APP135, Appnote135, Appnote 135