Application Note 162

INTERFACING THE DS18x20/DS1822 1-WIRE TEMPERATURE SENSOR IN A MICRO-CONTROLLER ENVIRONMENT

www.maxim-ic.com

INTRODUCTION

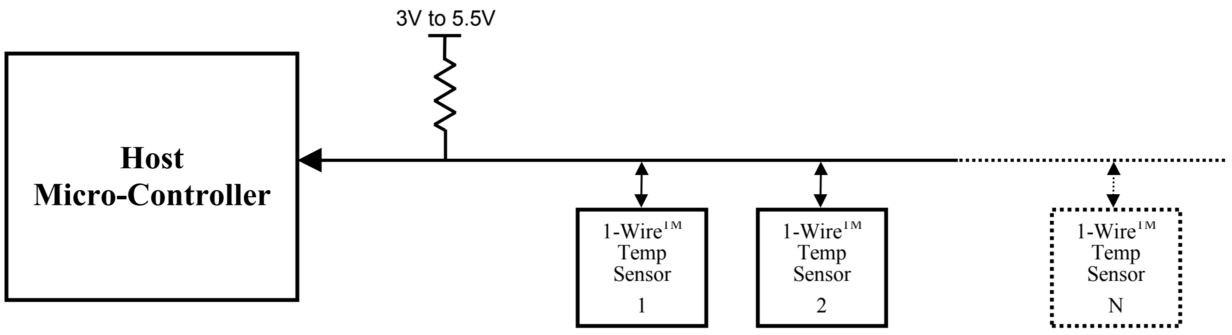
There are several methods available for interfacing 1-Wire™ devices such as the DS18B20, DS18S20 or DS1822 to a microcontroller. These methods range from simple software solutions, to using a Serial Interface chip such as the DS2480, to incorporating Dallas Semiconductor's VHDL 1-Wire ™ Master Controller in a custom ASIC. This article introduces the user to the simplest possible software solution for basic 1-Wire™ communication between a microcontroller and any number of DS18x20 or DS1822 temperature sensors.

Detailed timing and operational information for the DS18B20, DS18S20 and DS1822 is available in their respective datasheets, which can be obtained from the Maxim website.

HARDWARE CONFIGURATION

The block diagram in Figure 1 illustrates the simplicity of the hardware configuration when using multiple 1-Wire™ temperature sensors. A single-wire bus provides both communication access and power to all devices. Power to the bus is provided through the 4.7KΩ pullup resistor from a 3V to 5.5V supply rail. An almost unlimited number of 1-Wire™ devices can be connected to the bus because each device has a unique 64-bit ROM code identifier.

Figure 1. Host Micro-Controller Interface



INTERFACE TIMING

Communication with the DS18x20/DS1822 is achieved through the use of "time slots", which allow data to be transmitted over the 1-Wire ™ bus. Every communication cycle begins with a reset pulse from the micro-controller followed by a presence pulse from the DS18x20/DS1822 as shown in Figure 2.

A write time slot is initiated when the bus master pulls the 1-Wire™ bus from logic high (inactive) to logic low. All write time slots must be 60µs to 120µs in duration with a 1µs minimum recovery time between cycles. Write "0" and write "1" time slots are illustrated in Figure 3. During the write "0" time slot, the host micro-controller pulls the line low for the duration of the time slot. However, during the write "1" time slot, the micro-controller pulls the line low and then releases the line within 15µs after the start of the time slot.

A read time slot is initiated when the micro-controller pulls the bus low for 1µs then releases it so the DS18x20/DS1822 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 (see Figure 3).

Figure 2. Reset Pulse and Presence Pulse

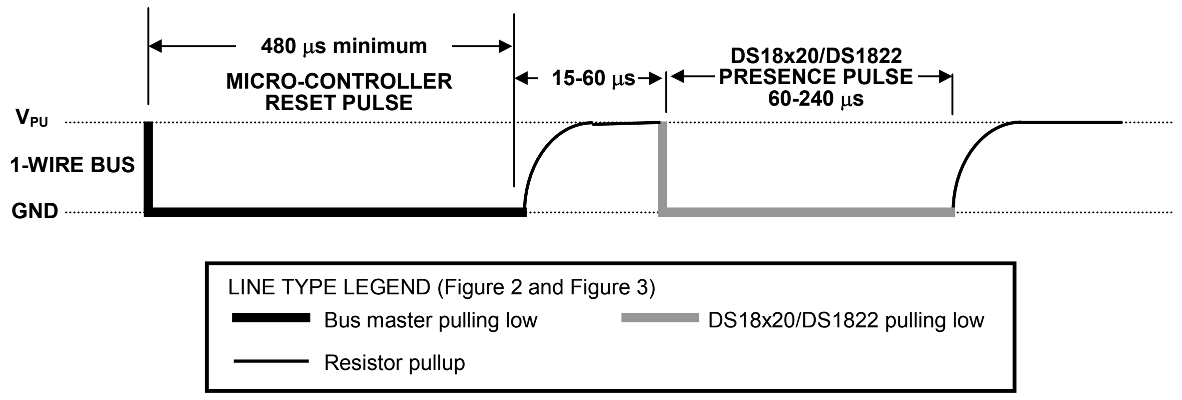
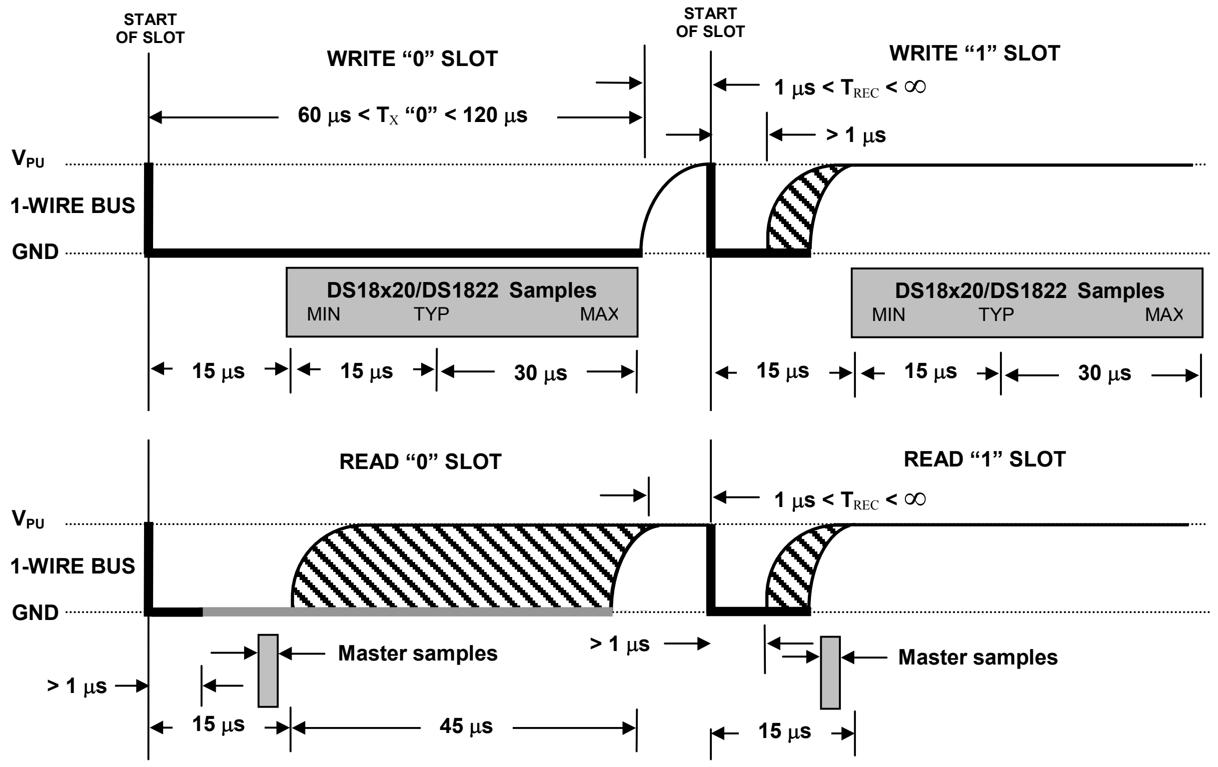

480 µs minimum
MICRO-CONTROLLER RESET PULSE
15-60 µs
DS18x20/DS1822 PRESENCE PULSE 60-240 µs
VPU
1-WIRE BUS
GND
LINE TYPE LEGEND (Figure 2 and Figure 3)
Bus master pulling low
DS18x20/DS1822 pulling low
Resistor pullup

Figure 3. Write and Read Time Slots



SOFTWARE CONTROL
In order to accurately control the special timing requirements of the 1-Wire^{TM} 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 is entirely dependent on the speed of the micro-controller. For the purpose of this article, the DS5000 (8051 compatible) micro-controller is used, which runs at 11.059MHz. The example to the right illustrates the "C" prototype function for creating the timing delay.
Delay Example

```
// 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 communication cycle must begin with a reset from the microcontroller, the "reset" function is the next most important function to be implemented. The reset time slot is 480us. By setting a delay of "3", followed by "25" (see the example below), the reset pulse will last for the required duration. Following the reset, the micro-controller must release so the DS18x20/DS1822 can indicate its "presence" by pulling the line low. Note that if multiple temperature sensors are on the bus, they will all respond simultaneously with a presence pulse.

Reset Example

```
//////////////////////////////////////////////////////////////////////////////
// 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 read and write function code segments shown in the following four examples provide the basic structure needed for all data bit and data byte read and write operations.

Read Bit Example

```
//////////////////////////////////////////////////////////////////////////////
// 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 Example

```
//////////////////////////////////////////////////////////////////////////////
// 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 Example

```
//////////////////////////////////////////////////////////////////////////////
// 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);
}
```

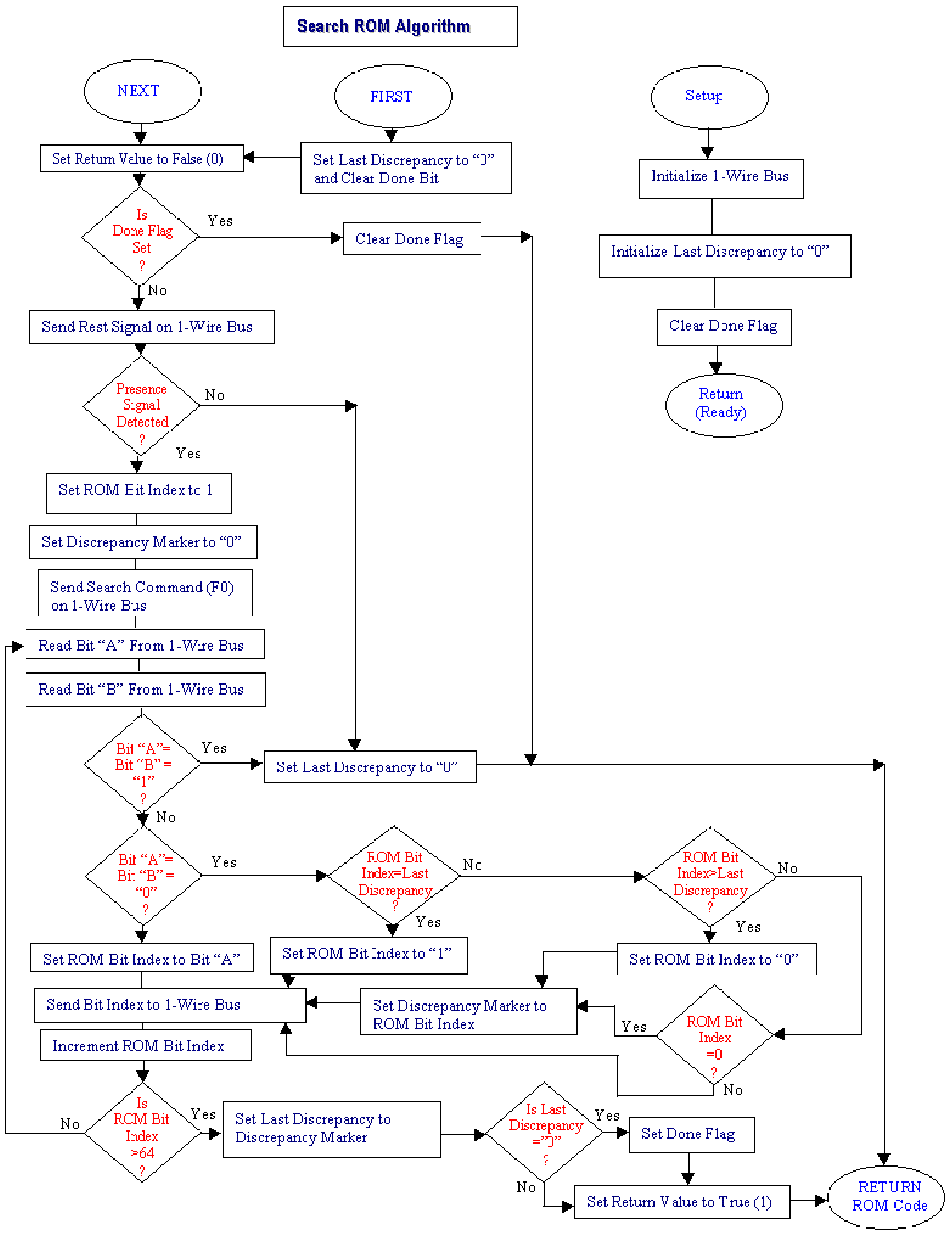
Write Byte Example

```
//////////////////////////////////////////////////////////////////////////////
// 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);

}
```

THE SEARCH ROM ALGORITHM

To take full advantage of the 1-Wire^{TM} net concept, the microcontroller must be able to communicate with any number of devices connected to the net. In order to do this, the microcontroller must learn the unique 64-bit ROM identification code for each device on the bus using the "Search ROM" algorithm illustrated in Figure 4. The example following Figure 4 explains a Search ROM routine for a bus with four slave devices. Sample code for a Search ROM routine is also shown. Once all the ROM codes have been identified, the "Match ROM" command can be used to communicate with any specific device on the net.

Figure 4. Search ROM Algorithm


Search ROM Algorithm
NEXT
FIRST
Setup
Set Return Value to False (0)
Set Last Discrepancy to "0" and Clear Done Bit
Initialize 1-Wire Bus
Is Done Flag Set ?
Yes
Clear Done Flag
Initialize Last Discrepancy to "0"
No
Send Rest Signal on 1-Wire Bus
Clear Done Flag
Return (Ready)
Presence Signal Detected ?
No
Yes
Set ROM Bit Index to 1
Set Discrepancy Marker to "0"
Send Search Command (F0) on 1-Wire Bus
Read Bit "A" From 1-Wire Bus
Read Bit "B" From 1-Wire Bus
Bit "A"= Bit "B" = "1" ?
Yes
Set Last Discrepancy to "0"
No
Bit "A"= Bit "B" = "0" ?
Yes
ROM Bit Index=Last Discrepancy ?
No
ROM Bit Index>Last Discrepancy ?
No
Yes
Yes
Set ROM Bit Index to Bit "A"
Set ROM Bit Index to "1"
Set ROM Bit Index to "0"
Send Bit Index to 1-Wire Bus
Set Discrepancy Marker to ROM Bit Index
Yes
ROM Bit Index =0 ?
Increment ROM Bit Index
No
No
Is ROM Bit Index >64 ?
Yes
Set Last Discrepancy to Discrepancy Marker
Is Last Discrepancy ="0" ?
Yes
Set Done Flag
No
Set Return Value to True (1)
RETURN ROM Code

ROM SEARCH EXAMPLE

During the ROM search process, the bus master must repeat a simple three-step routine: 1) read a ROM code bit from the slave devices, 2) read the complement of the bit, 3) write the selected value for that bit. The bus master must perform this three-step routine 64 times—once for each ROM code bit. After one complete pass, the bus master will know the ROM code for one slave device on the bus. The remaining devices and their ROM codes can be identified though additional passes.

The ROM Search process is illustrated by the following example that assumes four different devices are connected to the same 1-wire bus. The ROM codes of the four devices are as shown:

ROM1	00110101...
ROM2	10101010...
ROM3	11110101...
ROM4	00010001...

The search process goes as follows:

1. The bus master begins the initialization sequence by issuing a reset pulse. The slave devices respond by issuing simultaneous presence pulses.
2. The bus master then issues the Search ROM command on the 1-wire bus.
3. Each device will respond to the Search ROM command by placing the value of the first bit of their respective ROM codes onto the 1-wire bus. The master will then read the bus value. In this case, ROM1 and ROM4 will place a 0 onto the 1-wire bus, i.e., they will pull it low. ROM2 and ROM3 will place a 1 onto the 1-wire bus by allowing the line to stay high. The result is the logical AND of all devices on the line; therefore, the bus master will read a 0. All of the devices on the 1-wire bus will respond to this read by placing the complement of the first bit of their ROM codes onto the 1-wire bus: ROM1 and ROM4 will place a 1 onto the 1-wire bus, allowing the line to stay high, and ROM2 and ROM3 will place a 0 onto the bus, pulling it low. The bus master will now read the bus again and will again read a 0.

   Depending on the slave device ROM codes, there are four possible data combinations that the bus master can obtain from the two reads. These combinations can be interpreted as follows:

00	There are devices connected to the bus which have conflicting bits in the current ROM code bit position.
01	All devices connected to the bus have a 0 in this bit position.
10	All devices connected to the bus have a 1 in this bit position.
11	There are no devices connected to the 1-wire bus.

   In this example, bus master has read a 0 during each read, which tells it that there are some devices on the 1-wire bus that have a 0 in the first ROM code position and others that have a 1.
4. In response to the previous data, the bus master writes a 0 onto the bus. This deselects ROM2 and ROM3 for the remainder of this search pass, leaving only ROM1 and ROM4 "connected" to the 1-wire bus.
5. The bus master performs two more reads and receives a 0 followed by a 1. This indicates that all devices still connected to the bus have 0s as their second ROM data bit.
6. The bus master then writes a 0 to keep both ROM1 and ROM4 connected to the bus.
7. The bus master again executes two reads and receives two 0s. This indicates to the master that one of the devices on the 1-wire bus has a 0 in the third ROM code position and the other has a 1.
8. The bus master writes a 0 onto the bus, which deselects ROM1 and leaves ROM4 as the only device still connected.
9. The bus master reads the remainder of the ROM bits from ROM4 and continues to access the ROM4 device if desired. This completes the first ROM search pass; the bus master has now uniquely identified one slave (ROM4) on the 1-wire bus by learning its ROM code.
10. The bus master starts a new ROM search sequence by repeating steps 1 through 7.
11. The bus master now writes a 1 onto the bus (instead of a 0, as was done in step 8). This decouples ROM4, leaving only ROM1 still connected.
12. The bus master now reads the remainder of the ROM bits from ROM1 and can communicate with the ROM1 device if desired. This completes the second ROM search pass, and the master has now identified another slave device (ROM1).

13. The bus master starts a new ROM search by repeating steps 1 through 3.
14. The bus master now writes a 1 onto the bus (instead of a 0, as was done in step 4). This deselects ROM1 and ROM4 for the remainder of this search pass, leaving only ROM2 and ROM3 coupled to the bus.
15. The bus master executes two reads and receives two 0s.
16. The bus master writes a 0 onto the bus, which decouples ROM3, leaving only ROM2 connected to the bus.
17. The bus master reads the remainder of the ROM bits from ROM2 and communicates with the ROM2 device if desired. This completes the third ROM search pass, and the master has now identified the ROM2 slave device.
18. The bus master starts a fourth and final ROM search by repeating steps 13 through 15.
19. The bus master writes a 1 onto the bus (instead of a 0, as was done in step 16), which decouples ROM2, leaving only ROM3 connected to the bus.
20. The bus master reads the remainder of the ROM bits from ROM3 and communicates with the ROM3 device if desired. This completes the fourth ROM search pass, during which the master identified the ROM3 device. At this point the master has identified all the slave devices on the bus, and from this point on the bus master can individually address any of the devices using their ROM codes.

Note**:** The bus master learns the unique ROM code of one 1-wire device during each ROM search pass. The time required to learn one ROM code is:

$$960\ \mu s + (8 + 3 \times 64)\ 61\ \mu s = 13.16\ m$$

The bus master is therefore capable of identifying 75 different 1-wire slave devices per second.

SEARCH ROM CODE EXAMPLES

As shown in the prototype function below, the "Find Devices" function begins with a 1-Wire reset to determine if any devices are on the net, and if so, to wake them up. The "First" function is then called (see the next page), to keep track of the discrepancy bits and return to "Next", which finds each unique device on the net.

The "Next" function is quite extensive and does most of the work in finding each unique 64-bit ROM code identifier for each device on the net.

```
// FIND DEVICES
void FindDevices(void)
{
          unsigned char m;
          if(!ow_reset())                               //Begins when a presence is detected
          {
                    if(First())                         //Begins when at least one part is found
                    {
                              numROMs=0;
          do
                              {
                                        numROMs++;
                                        for(m=0;m<8;m++)
                                        {
                                                  FoundROM[numROMs][m]=ROM[m];  //Identifies ROM
                                                                      \\number on found device
                                        }         printf("\nROM CODE =%02X%02X%02X%02X\n",
                                                  FoundROM[5][7],FoundROM[5][6],FoundROM[5][5],FoundROM[5][4],
                                                  FoundROM[5][3],FoundROM[5][2],FoundROM[5][1],FoundROM[5][0]);
                              }while (Next()&&(numROMs<10)); //Continues until no additional devices are found
                    }
   }

}
```

```
// FIRST
//   The First function resets the current state of a ROM search and calls
//   Next to find the first device on the 1-wire bus.
//
unsigned char First(void)
{
        lastDiscrep  = 0;                    // reset the rom search last discrepancy global
        doneFlag = FALSE;
        return Next();                           // call Next and return its return value
}
```

```
// NEXT
//  The Next function searches for the next device on the 1-wire bus. If
//  there are no more devices on the 1-wire then false is returned.
//

unsigned char Next(void)
{
        unsigned char m = 1;                    // ROM Bit index
        unsigned char n = 0;                    // ROM Byte index
        unsigned char k = 1;                    // bit mask
        unsigned char x = 0;
        unsigned char discrepMarker = 0;   // discrepancy marker
        unsigned char g;                        // Output bit
        unsigned char nxt;                              // return value
        int      flag;


        nxt = FALSE;                                    // set the next flag to false
        dowcrc = 0;                                     // reset the dowcrc

        flag = ow_reset();                      // reset the 1-wire
        if(flag||doneFlag)                      // no parts -> return false
        {
                lastDiscrep = 0;                // reset the search
                return FALSE;
        }

        write_byte(0xF0);                               // send SearchROM command

        do
                                                                // for all eight bytes
        {
                x = 0;
                if(read_bit()==1) x = 2;
                delay(6);
                if(read_bit()==1) x |= 1;    // and its complement
                if(x ==3)                               // there are no devices on the 1-wire
                        break;
```

The "Next" function is continued on the following page.

```
        else
                {
                if(x>0)                         // all devices coupled have 0 or 1
                    g = x>>1;                       // bit write value for search
                else
                {
                                                // if this discrepancy is before the last
                                                //  discrepancy on a previous Next then pick
                                                //  the same as last time
                    if(m<lastDiscrep)
                        g = ((ROM[n]&k)>0);
                    else                                    // if equal to last pick 1
                        g = (m==lastDiscrep);       // if not then pick 0

                                                    // if 0 was picked then record
                                                    //      position with mask k
                        if (g==0) discrepMarker = m;
                }

                if(g==1)                        // isolate bit in ROM[n] with mask k
                    ROM[n] |= k;
                else
                    ROM[n] &= ~k;
                write_bit(g);                   // ROM search write

                m++;                                // increment bit counter m
                k = k<<1;                           // and shift the bit mask k

                if(k==0)                        // if the mask is 0 then go to new ROM

                {               // byte n and reset mask

                    ow_crc(ROM[n]);             // accumulate the CRC
                    n++; k++;
                }
            }
        }while(n<8);                            //loop until through all ROM bytes 0-7
        if(m<65||dowcrc)                    // if search was unsuccessful then
            lastDiscrep=0;                      // reset the last discrepancy to 0
        else
        {
                                                // search was successful, so set lastDiscrep,
                                                //      lastOne, nxt
            lastDiscrep = discrepMarker;
            doneFlag = (lastDiscrep==0);
            nxt = TRUE;             // indicates search is not complete yet, more
                                                    //      parts remain
        }


        return nxt;
}
```

PERFORMING A CYCLIC REDUNDANCY CHECK

A cyclic redundancy check (CRC) can be accomplished using the functions shown below and should be included when performing the Search ROM function.

```
//////////////////////////////////////////////////////////////////////////////
// ONE WIRE CRC
//
unsigned char ow_crc( unsigned char x)
{
   dowcrc = dscrc_table[dowcrc^x];
           return dowcrc;
}
```

```
#define  FALSE  0
#define TRUE     1


//////////////////////////////////////////////////////////////////////////////
// GLOBAL VARIABLES
//
unsigned char ROM[8];                              // ROM Bit
unsigned char lastDiscrep = 0;                 // last discrepancy
unsigned char doneFlag = 0;                        // Done flag
unsigned char FoundROM[5][8];               // table of found ROM codes
unsigned char numROMs;

unsigned char dowcrc;
unsigned char code dscrc_table[] = {
        0, 94,188,226, 97, 63,221,131,194,156,126, 32,163,253, 31, 65,
      157,195, 33,127,252,162, 64, 30, 95,  1,227,189, 62, 96,130,220,
       35,125,159,193, 66, 28,254,160,225,191, 93,  3,128,222, 60, 98,
      190,224,  2, 92,223,129, 99, 61,124, 34,192,158, 29, 67,161,255,
       70, 24,250,164, 39,121,155,197,132,218, 56,102,229,187, 89,  7,
      219,133,103, 57,186,228,  6, 88, 25, 71,165,251,120, 38,196,154,
      101, 59,217,135,  4, 90,184,230,167,249, 27, 69,198,152,122, 36,
      248,166, 68, 26,153,199, 37,123, 58,100,134,216, 91,  5,231,185,
      140,210, 48,110,237,179, 81, 15, 78, 16,242,172, 47,113,147,205,
       17, 79,173,243,112, 46,204,146,211,141,111, 49,178,236, 14, 80,
      175,241, 19, 77,206,144,114, 44,109, 51,209,143, 12, 82,176,238,
       50,108,142,208, 83, 13,239,177,240,174, 76, 18,145,207, 45,115,
      202,148,118, 40,171,245, 23, 73,  8, 86,180,234,105, 55,213,139,
       87,  9,235,181, 54,104,138,212,149,203, 41,119,244,170, 72, 22,
      233,183, 85, 11,136,214, 52,106, 43,117,151,201, 74, 20,246,168,
      116, 42,200,150, 21, 75,169,247,182,232, 10, 84,215,137,107, 53};
```

READING DEVICE TEMPERATURE

If there is a single device on the net, then the “Read Temperature” function can be used directly as shown below. However, if multiple devices are on the net, in order to avoid data collisions, the “Match ROM” function must be used to select a specific device.

The code example below was written specifically for use with the DS18S20 temperature sensor. To use this code with the DS18B20 or DS1822, it must be modified slightly due to differences in the temperature register format. Refer to the respective datasheet for temperature register format information.

```
void Read_Temperature(void)
{
char get[10];
char temp_lsb,temp_msb;
int k;
char temp_f,temp_c;


        ow_reset();
        write_byte(0xCC);  //Skip ROM
        write_byte(0x44);  // Start Conversion

        delay(5);
        ow_reset();
        write_byte(0xCC); // Skip ROM
        write_byte(0xBE);  // Read Scratch Pad
        for (k=0;k<9;k++){get[k]=read_byte();}

printf("\n ScratchPAD DATA = %X%X%X%X%X\n",get[8],get[7],get[6],get[5],get[4],get[3],get[2],get[1],get[0]);

        temp_msb = get[1]; // Sign byte + lsbit
        temp_lsb = get[0]; // Temp data plus lsb
                if (temp_msb <= 0x80){temp_lsb = (temp_lsb/2);} // shift to get whole degree
        temp_msb = temp_msb & 0x80; // mask all but the sign bit
                if (temp_msb >= 0x80) {temp_lsb = (~temp_lsb)+1;} // twos complement
                if (temp_msb >= 0x80) {temp_lsb = (temp_lsb/2);}// shift to get whole degree
                if (temp_msb >= 0x80) {temp_lsb = ((-1)*temp_lsb);} // add sign bit
printf( "\nTempC=  %d degrees C\n", (int)temp_lsb );  // print temp. C
        temp_c = temp_lsb; // ready for conversion to Fahrenheit
        temp_f = (((int)temp_c)* 9)/5 + 32;
printf( "\nTempF=  %d degrees F\n", (int)temp_f );  // print temp. F
}
```

READING THE SCRATCH PAD MEMORY

The Scratch Pad memory provides the user with all the necessary device data including temperature, TH and TL programmable thermometer settings, as well as the Count Remain and Count Per C data used in fractional temperature measurements. The CRC byte is also included in Scratch Pad memory.

```
void Read_ScratchPad(void)
{

int j;
char pad[10];
                    printf("\nReading ScratchPad Data\n");
                    write_byte(0xBE);
                              for (j=0;j<9;j++){pad[j]=read_byte();}
printf("\n ScratchPAD DATA =
%X%X%X%X%X%X\n",pad[8],pad[7],pad[6],pad[5],pad[4],pad[3],pad[2],pad[1],pad[0]);
}
```

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.

```
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]);
}
```

The “Match ROM” function must provide the 64-bit ROM-ID to select an individual device on the net.

```
// Perform Match ROM
//
unsigned char Send_MatchRom(void)
{
          unsigned char i;


          if(ow_reset()) return false;
          write_byte(0x55);                                          // match ROM
          for(i=0;i<8;i++)
          {
                    write_byte(FoundROM[numROMs][i]);       //send ROM code
          }

          return true;
}
```

Appendix A
DS5000 (8051 Source Code)

```
// 1wiretalk.c -- Functions for the Dallas Semiconductor DS18x20/DS1822
// Two-Wire Temperature Sensor
//  Designed for 8051 microcontrollers
// This code was developed using the DS5000/DS2251T
// Please note that 128K RAM size is required to run this program.
/*---------------------------------------------------------------------*/
//#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     */
/*---------------------------------------------------------------------*/
/*--------------------------------------------------------------------* /

////////////////////////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 DS1820 Temperature Reading and\n");
printf ("           Search ROM code.\n");
printf ("           Updated  Code August, 2001 \n");
printf ("           [C Program for DS500x 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. Perform Search ROM\n");
printf ("           4. Read Scratch PAD\n");
printf ("           5. Read Temperature\n");
printf ("           6. Find All Devices\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();
                                                Read_ROMCode();

                        case '3': printf("\n          3. Performing Search ROM\n");
                                               ow_reset();
                                               First();
        printf("\nROM CODE =%02X%02X%02X%02X\n",

        FoundROM[5][7],FoundROM[5][6],FoundROM[5][5],FoundROM[5][4],

        FoundROM[5][3],FoundROM[5][2],FoundROM[5][1],FoundROM[5][0]);
                         break;

                        case '4': printf ("\n         4. Read Scratch PAD\n");
                                                 ow_reset();
                                                 write_byte(0xCC);  // Skip ROM
                                                 Read_ScratchPad();
                                               break;

                        case '5': printf ("\n         5. Read Temperature\n");
                                                 Read_Temperature();   //initiates a temperature reading
                                               break;


                        case '6': printf ("\n         6. Find All Devices\n");
                                                 ow_reset();
                                                FindDevices();
                                               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
```