应用笔记 187

1-WIRE搜索算法

摘要 : *Maxim*的*1-Wire*®器件都带有一个*64*位的唯一注册码，存储在只读存储器内*(ROM)*，能够在*1-Wire*网络中通过*1-Wire*主机对其寻址。如果*1-Wire*网络中从机器件的*ROM*码是未知的，则可采用搜索算法查找这些码。本文详细说明了搜索算法的原理，并提供了一个范例，便于用户使用。该算法对任何现有的或将要推出的*1-Wire*器件都是有效的。

绪论

Maxim的每片1-Wire器件都有唯一的64位注册码，它存储在只读存储器(ROM)中。在1-Wire网络中，注册码用于1-Wire主机对从机器件进行逐一寻址。如果1-Wire网络中从机器件的ROM码是未知的，可以通过搜索算法来找到此码。本文不仅详细地解释了搜索算法，而且 还提供了实现快速整合的例程。该算法适用于任何具有1-Wire接口特性的现有产品及未来产品。

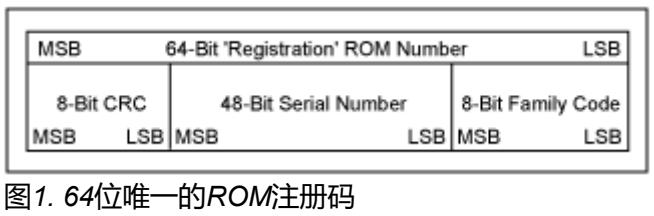


图*1. 64*位唯一的*ROM*注册码

搜索算法

搜索算法采用的是二叉树型结构，搜索过程沿各分节点进行，直到找到器件的ROM码即叶子为 止；后续的搜索操作沿着节点上的其它路径进行，按照同样的方式直到找到总线上的所有器件代 码。

搜索算法首先通过复位(reset)和在线应答脉冲(presence pulse)时隙将1-Wire总线上的所有器 件复位；成功地执行该操作后，发送1 个字节的搜索命令；搜索命令使1-Wire器件准备就绪、开 始进行搜索操作。

搜索命令分为两类：标准搜索命令(F0 hex)用来搜索连接到网络中所有器件；报警或有条件搜索命 令(EC hex)只用来搜索那些处于报警状态下的器件，这种方式缩小了搜索范围，可以快速查找到所 需要注意的器件。

搜索命令发出之后，开始实际的搜索过程。首先总线上的所有从机器件同时发送ROM码(也叫 注册码)中的第一位(最低有效位) (参见**图1**)。与所有的1-Wire通信一样，无论是读取数据 还是向从机器件写数据，都由1-Wire主机启动每一位操作。按照1-Wire的特性，当所有从机器 件同时应答主机时，结果相当于全部发送数据位的逻辑AND；从机发送其ROM码的第一位后， 主机启动下一位操作、接着从机发送第一位数据的补码；从两次读到的数据位可以对ROM码的 第一位做出几种判断(参见**表1**)。

表1. 检索信息位

Bit (true)	Bit (complement)	Information Known
0	0	There are both 0s and 1s in the current bit position of the participating ROM numbers. This is a discrepancy.
0	1	There are only 0s in the bit of the participating ROM numbers.
1	0	There are only 1s in the bit of the participating ROM numbers.
1	1	No devices participating in search.

按照搜索算法的要求，1-Wire主机必须向总线上的从机发回一个指定位；如果从机器件中ROM码的当前位的值与该数据位匹配，则继续参与搜索过程；若从机器件的当前位与之不匹配，则该 器件转换到等待状态，并保持等待状态直到下一个1-Wire复位信号到来。其余63位ROM码的搜 索依然按照这种'读两位'、'写一位'的模式进行重复操作(参见**表2**)。按照这种搜索算法 进行下去，最终除了一个从机器件外所有从机将进入等待状态，经过最后一轮检测，就可得到最 后保留(未进入等待状态)器件的ROM码。在后续搜索过程中，选用不同的路径(或分支)来 查找其它器件的ROM码。需要注意的是本文ROM码的数据位用第1位(最低有效位)到第64 位(最高有效位)表示，而不是我们常用的那种第0位到第63位的模式；这样设置允许将差异位 置计数器初始值置为0，为以后的比较提供了方便。

表2. 1-Wire主机和从机的搜索过程

Master	Slave
1-Wire reset stimulus	Produce presence pulse

Write search command (normal or alarm)	Each slave readies for search.
Read 'AND' of bit 1	Each slave sends bit 1 of its ROM number.
Read 'AND' of complement bit 1	Each slave sends complement bit 1 of its ROM number.
Write bit 1 direction (according to algorithm)	Each slave receives the bit written by master, if bit read is not the same as bit 1 of its ROM number then go into a wait state.
Read 'AND' of bit 64	Each slave sends bit 64 of its ROM number.
Read 'AND' of complement bit 64	Each slave sends complement bit 64 of its ROM number.
Write bit 64 direction (according to algorithm)	Each slave receives the bit written by master, if bit read is not the same as bit 64 of its ROM number then go into a wait state.

从表1可以看出：如果所有总线上的器件在当前位具有相同值，那么只有一条分支路径可选；总 线上没有器件响应的情况是一种异常状态，可能是要查找的器件在搜寻过程中与1-Wire总线脱 离。如果出现这种情况，应中止搜索，并发出1-Wire复位信号起始新的搜索过程。如果当前位既有0也有1，这种情况称为位值差异，它对在后续搜索过程中查找器件起关键 作用。搜索算法指定在第一轮查询中若出现差异(数据位/补码 = 0/0)，则选用‘0’路径。注 意：这一点是由本文档中介绍的特定算法决定的，其它算法中或许首先选用‘1’路径。记录最 后一次值差异的位置以供下一次搜索使用，**表3**列出了出现值差异时路径的选取情况。

表3. 搜索路径方向

Search Bit Position vs Last Discrepancy	Path Taken
=	take the '1' path
<	take the same path as last time (from last ROM number found)
>	take the '0' path

搜索算法计算还对最初8位过程中出现的最后一次位差异保持跟踪；64位注册码的前8位是家族 码，在器件的搜索过程中可以按照其家族码进行分类。记录家族码的最后一次差异可以用于有选 择性地跳过1-Wire器件的整个分组。如需进行选择性的搜索，可参考关于高级变量搜索的详细解 释。64位ROM码中包含8位循环冗余校验码(CRC)；CRC值用于验证是否搜索到正确的ROM码。ROM码的排列如图1所示。

DS2480B系列串口到1-Wire线路的驱动程序在硬件中实现了部分与本文档中相同的搜索算法；详 细资料请参阅DS2480B数据资料和应用笔记192，*DS2480B*串行接口*1-Wire*线驱动器的使用的详细介绍；从DS2490 USB口到1-Wire桥接器硬件电路 中实现了整个搜索过程。

图2列出了对一个从器件进行搜索的流程图；注意：右侧*Reference*栏对在流程图中出现的符号进行了 说明；在本文档的源代码附录中也将用到这些专用符号。

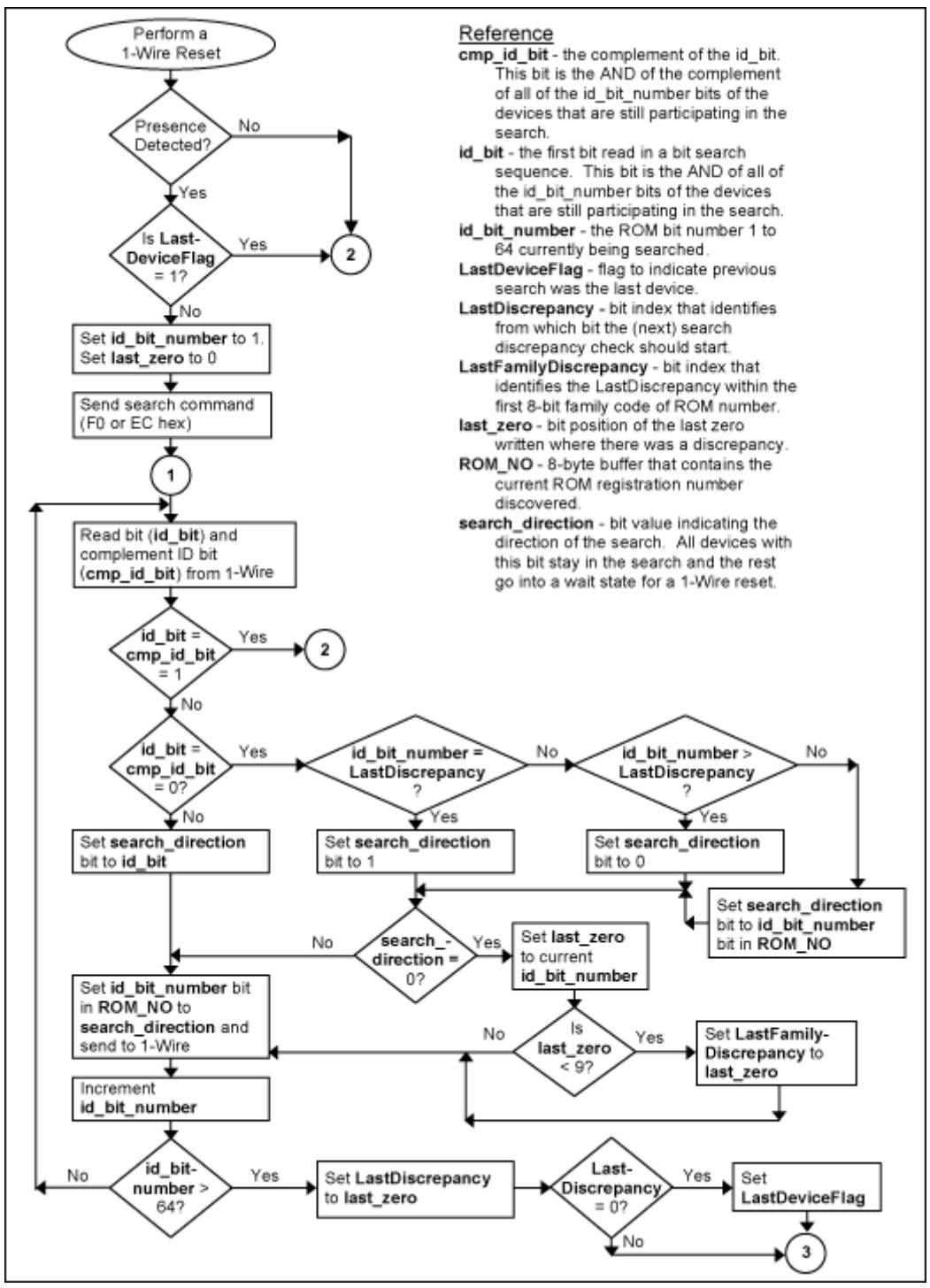


图2. 搜索流程

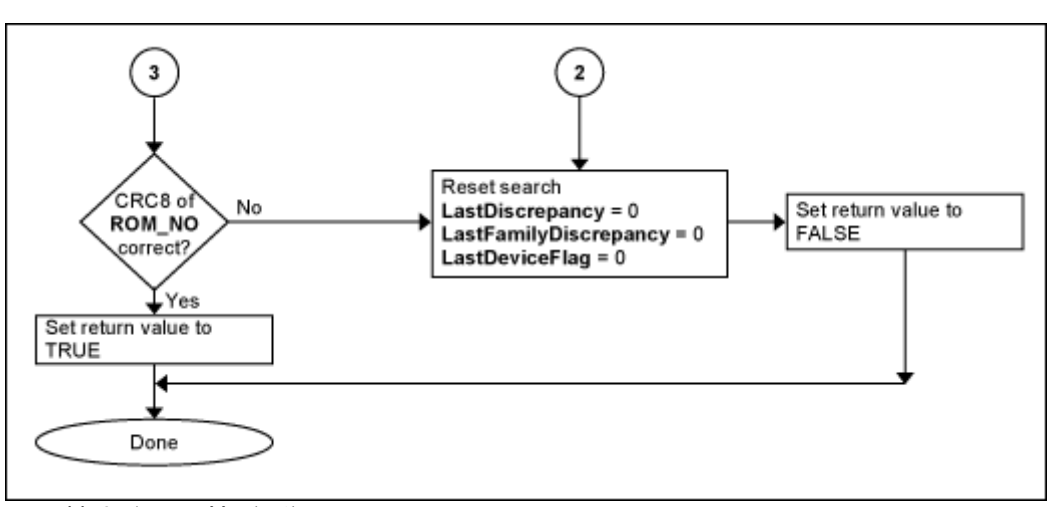


图2. 搜索流程，第2部分

搜索算法通过对LastDiscrepancy、LastFamilyDiscrepancy、LastDeviceFlag和ROM_NO值(参见**表4**)的处理，利用上述流程实现了两个不同类型的搜索操作；这两个操作是搜索1-Wire器件ROM码的基础。

First

‘FIRST’操作是搜索1-Wire总线上的第一个从机器件。该操作是通过将LastDiscrepancy、 LastFamilyDiscrepancy和LastDeviceFlag置零，然后进行搜索完成的。最后ROM码从ROM_NO寄存器中读出。若1-Wire总线上没有器件，复位序列就检测不到应答脉冲，搜索过程中止。

Next

‘NEXT’操作是搜索1-Wire总线上的下一个从机器件；一般情况下，此搜索操作是在‘FIRST’操作之后或上一次‘NEXT’操作之后进行；保持上次搜索后这些值的状态不变、执行又一次搜 索即可实现‘NEXT’操作。之后从ROM_NO寄存器中来读出新一个ROM码。若前一次搜索到 的是1-Wire上的最后一个器件，则返回一个无效标记FALSE，并且把状态设置成下一次调用搜 索算法时将是‘FIRST’操作的状态。

图3 (a, b, c)例举了三个器件的搜索过程，为便于说明，设器件的ROM码只有2位。

Devices

A = 01 (binary: bit 2, bit 1)
B = 00
C = 11

FIRST

AND result of 'true' bit read

AND result of the 'complement' bit read

Bit written by master, path taken

bit 1	Read bit	Read complement-bit	Write direction	
A	1	0		
B	0	1		
C	1	0		
	0	0	0	(bit position > LastDiscrepancy)

bit 2	Read bit	Read complement-bit	Write direction	
A	(wait state)			
B	0	1		
C	(wait state)			
	0	1	0	(only one path available)

Device B is found 00, LastDiscrepancy is now 1

NEXT

bit 1	Read bit	Read complement-bit	Write direction	
A	1	0		
B	0	1		
C	1	0		
	0	0	1	(bit position = LastDiscrepancy)

bit 2	Read bit	Read complement-bit	Write direction	
A	0	1		
B	(wait state)			
C	1	0		
	0	0	0	(bit position > LastDiscrepancy)

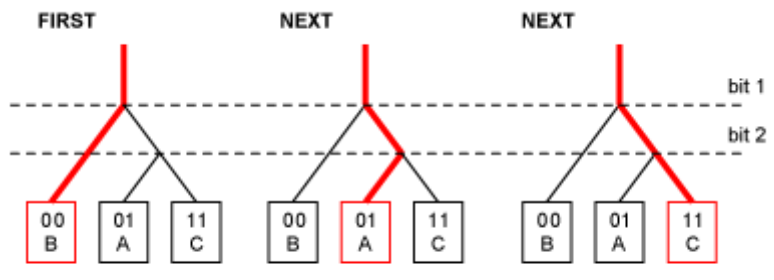
Device A is found 01, LastDiscrepancy is now 2

NEXT

bit 1	Read bit	Read complement-bit	Write direction	
A	1	0		
B	0	1		
C	1	0	(bit position < LastDiscrepancy)	
	0	0	1	

bit 2	Read bit	Read complement-bit	Write direction	
A	0	1		
B	(wait state)			
C	1	0	(bit position = LastDiscrepancy)	
	0	0	1	

Device C is found 11, LastDeviceFlag is TRUE



Note: Each branching at a bit level denotes a 'discrepancy' where both the bit and complement-bit return '0'.

(for simplicity the family discrepancy register and tracking has been left out of this example)

FIRST
- **LastDiscrepancy** = **LastDeviceFlag** = 0
- Do 1-Wire reset and wait for presence pulse, if no presence pulse then done
- **id_bit_number** = 1, **last_zero** = 0
- Send search command, 0F hex
- Read first bit **id_bit**: 1 (Device A) AND 0 (Device B) AND 1 (Device C) = 0
- Read complement of first bit **cmp_id_bit**: 0 (Device A) AND 1 (Device B) AND 0 (Device C) = 0
- Since **id_bit_number** > **LastDiscrepancy** then **search_direction** = 0, **last_zero** = 1
- Send **search_direction** bit of 0, both Devices A and C go into wait state
- Increment **id_bit_number** to 2
- Read second bit **id_bit**: 0 (Device B) = 0
- Read complement of second bit **cmp_id_bit**: 1 (Device B) = 1
- Since bit and complement are different then **search_direction** = **id_bit**
- Send **search_direction** bit of 0, Device B is discovered with **ROM_NO** of '00' and is now selected
- **LastDiscrepancy** = **last_zero**

NEXT
- Do 1-Wire reset and wait for presence pulse, if no presence pulse then done
- **id_bit_number** = 1, **last_zero** = 0
- Send search command, 0F hex
- Read first bit **id_bit**: 1 (Device A) AND 0 (Device B) AND 1 (Device C) = 0
- Read complement of first bit **cmp_id_bit**: 0 (Device A) AND 1 (Device B) AND 0 (Device C) = 0
- Since **id_bit_number** = **LastDiscrepancy** then **search_direction** = 1
- Send **search_direction** bit of 1, Devices B goes into wait state
- Increment **id_bit_number** to 2
- Read second bit **id_bit**: 0 (Device A) AND 1 (Device C) = 0
- Read complement of second bit **cmp_id_bit**: 1 (Device A) AND 0 (Device C) = 0
- Since **id_bit_number** > **LastDiscrepancy** then **search_direction** = 0, **last_zero** = 2
- Send **search_direction** bit of 0, Devices C goes into wait state
- Device A is discovered with **ROM_NO** of '01' and is now selected
- **LastDiscrepancy** = **last_zero**

NEXT
- Do 1-Wire reset and wait for presence pulse, if no presence pulse then done
- **id_bit_number** = 1, **last_zero** = 0
- Send search command, 0F hex
- Read first bit **id_bit**: 1 (Device A) AND 0 (Device B) AND 1 (Device C) = 0
- Read complement of first bit **cmp_id_bit**: 0 (Device A) AND 1 (Device B) AND 0 (Device C) = 0
- Since **id_bit_number** < **LastDiscrepancy** then **search_direction** = **ROM_NO** (first bit) = 1
- Send **search_direction** bit of 1, Devices B goes into wait state
- Increment **id_bit_number** to 2
- Read second bit **id_bit**: 0 (Device A) AND 1 (Device C) = 0
- Read complement of second bit **cmp_id_bit**: 1 (Device A) AND 0 (Device C) = 0
- Since **id_bit_number** = **LastDiscrepancy** then **search_direction** = 1
- Send **search_direction** bit of 1, Devices A goes into wait state
- Device C is discovered with **ROM_NO** of '11' and is now selected
- **LastDiscrepancy** = **last_zero** which is 0 so **LastDeviceFlag** = TRUE

NEXT
- **LastDeviceFlag** is true so return FALSE
- **LastDiscrepancy** = **LastDeviceFlag** = 0

图*3.* 搜索过程举例

高级变量搜索

有3种利用同一组状态变量LastDiscrepancy、LastFamilyDiscrepancy、LastDeviceFlag、ROM_NO实现的高级变量搜索算法，这几种高级搜索算法允许来指定作为搜索 目标或需要跳过搜索的器件的类型(家族码)以及验证某类型的器件是否在线(参见**表4**)。

Verify

'VERIFY'操作用来检验已知ROM码的器件是否连接在1-Wire总线上，通过提供ROM码并对 该码进行目标搜索就可确定此器件是否在线。首先，将ROM_NO寄存器值设置为已知的ROM码 值，然后将LastDiscrepancy和LastDeviceFlag标志位分别设置为64 (40h)和0； 进行搜索操作， 然后读ROM_NO的输出结果；如果搜索成功并且ROM_NO中存储的仍是要搜索器件的ROM码 值，那么此器件就在1-Wire总线上。

Target Setup

'TARGET SETUP'操作就是用预置搜索状态的方式首先查找一个特殊的家族类型，每个1-Wire器件都有一个字节的家族码内嵌在ROM码中(参见图1)，主机可以通过家族码来识别器件所具 有的特性和功能。若1-Wire总线上有多片器件时，通常是将搜索目标首先定位在需注意的器件类 型上。为了将一个特殊的家族作为搜索目标，需要将所希望的家族码字节放到ROM_NO寄存器 的第一个字节中，并且将ROM_NO寄存器的复位状态置零，然后将LastDiscrepancy设置为64 (40h)； 把LastDeviceFlag和LastFamilyDiscrepancy设置为0。在执行下一次搜索算法时就能找出 所期望的产品类型的第一个器件；并将此值存入ROM_NO寄存器。需要注意的是如果1-Wire总 线上没有挂接所期望的产品类型的器件，就会找出另一类型的器件，所以每次搜索完成后，都要 对ROM_NO寄存器中存储的结果进行校验。

Family Skip Setup

'FAMILY SKIP SETUP'操作用来设置搜索状态以便跳过搜索到的指定家族中的所有器件，此操 作只有在一个搜索过程结束后才能使用。通过把LastFamilyDiscrepancy复制到LastDiscrepancy， 并清除LastDeviceFlag即可实现该操作；在下一搜索过程就会找到指定家族中的下一个器件。如 果当前家族码分组是搜索过程中的最后一组，那么搜索过程结束并将LastDeviceFlag置位。

表4. 搜索变量状态的设置

	LastDiscrepancy	LastFamily- Discrepancy	LastDeviceFlag	ROM_NO
FIRST	0	0	0	result
NEXT	leave unchanged	leave unchanged	leave unchanged	result
VERIFY	64	0	0	set with ROM to verify, check if same after search
TARGET SETUP	64	0	0	set first byte to family code, set rest to zeros

FAMILY SKIP SETUP	copy from LastFamilyDiscrepancy	0	0	leave unchanged

结论

本文提供的搜索算法可以找出任意给定的1-Wire器件组中独一无二的ROM码，这是保证多点1-Wire总线应用的关键，已知ROM码后就可以对逐一选定的某个1-Wire器件来进行操作。本文还 对一些变量搜索算法做了详细论述，这些变量搜索算法能够查找或跳过特定类型的1-Wire器件。附录给出了实现搜索过程和所有变量搜索算法的例程，并给出了‘C’程序代码。

附录

下面给出了搜索算法的‘C’程序代码及实现每个变量搜索算法的函数。FamilySkipSetup和TargetSetup函数实际上并没有进行搜索操作，它们只不过是用来设置搜索寄存器，以便在下一次 执行‘NEXT’操作时能跳过或找到所期望的类型。需要注意的是低级1-Wire函数可通过调用TMEX API实现。这些程序调用只是用于系统测试，也可以用特定平台调用。关于TMEX API和其它一些1-Wire API的详细资料请参考应用笔记*155*。

下列TMEX API测试程序的源代码可从Maxim网页下载。

```
// TMEX API TEST BUILD DECLARATIONS
#define TMEXUTIL
#include "ibtmexcw.h"
long session_handle;
// END TMEX API TEST BUILD DECLARATIONS

// definitions
#define FALSE 0
#define TRUE  1

// method declarations
int  OWFirst();
int  OWNext();
int  OWVerify();
void OWTargetSetup(unsigned char family_code);
void OWFamilySkipSetup();
int  OWReset();
void OWWriteByte(unsigned char byte_value);
void OWWriteBit(unsigned char bit_value);
unsigned char OWReadBit();
int  OWSearch();
unsigned char docrc8(unsigned char value);

// global search state
unsigned char ROM_NO[8];
int LastDiscrepancy;
int LastFamilyDiscrepancy;
int LastDeviceFlag;
unsigned char crc8;

//--------------------------------------------------------------------------
// Find the 'first' devices on the 1-Wire bus
// Return TRUE  : device found, ROM number in ROM_NO buffer
//        FALSE : no device present
//
int OWFirst()
{
   // reset the search state
   LastDiscrepancy = 0;
   LastDeviceFlag = FALSE;
   LastFamilyDiscrepancy = 0;

   return OWSearch();
}

//--------------------------------------------------------------------------
// Find the 'next' devices on the 1-Wire bus
// Return TRUE  : device found, ROM number in ROM_NO buffer
//        FALSE : device not found, end of search
//
int OWNext()
{
   // leave the search state alone
   return OWSearch();
}

//--------------------------------------------------------------------------
// Perform the 1-Wire Search Algorithm on the 1-Wire bus using the existing
// search state.
// Return TRUE  : device found, ROM number in ROM_NO buffer
//        FALSE : device not found, end of search
//
int OWSearch()
{
   int id_bit_number;
   int last_zero, rom_byte_number, search_result;
   int id_bit, cmp_id_bit;
   unsigned char rom_byte_mask, search_direction;

   // initialize for search
   id_bit_number = 1;
   last_zero = 0;
   rom_byte_number = 0;
   rom_byte_mask = 1;
   search_result = 0;
   crc8 = 0;

   // if the last call was not the last one
   if (!LastDeviceFlag)
   {
      // 1-Wire reset
      if (!OWReset())
      {
```

```
   // reset the search
   LastDiscrepancy = 0;
   LastDeviceFlag = FALSE;
   LastFamilyDiscrepancy = 0;
   return FALSE;
}

// issue the search command
OWWriteByte(0xF0);

// loop to do the search
do
{
   // read a bit and its complement
   id_bit = OWReadBit();
   cmp_id_bit = OWReadBit();

   // check for no devices on 1-wire
   if ((id_bit == 1) && (cmp_id_bit == 1))
      break;
   else
   {
      // all devices coupled have 0 or 1
      if (id_bit != cmp_id_bit)
         search_direction = id_bit;  // bit write value for search
      else
      {
         // if this discrepancy if before the Last Discrepancy
         // on a previous next then pick the same as last time
         if (id_bit_number < LastDiscrepancy)
            search_direction = ((ROM_NO[rom_byte_number] & rom_byte_mask) > 0);
         else
            // if equal to last pick 1, if not then pick 0
            search_direction = (id_bit_number == LastDiscrepancy);

         // if 0 was picked then record its position in LastZero
         if (search_direction == 0)
         {
            last_zero = id_bit_number;

            // check for Last discrepancy in family
            if (last_zero < 9)
               LastFamilyDiscrepancy = last_zero;
         }
      }

      // set or clear the bit in the ROM byte rom_byte_number
      // with mask rom_byte_mask
      if (search_direction == 1)
        ROM_NO[rom_byte_number] |= rom_byte_mask;
      else
        ROM_NO[rom_byte_number] &= ~rom_byte_mask;

      // serial number search direction write bit
      OWWriteBit(search_direction);

      // increment the byte counter id_bit_number
      // and shift the mask rom_byte_mask
      id_bit_number++;
      rom_byte_mask <<= 1;

      // if the mask is 0 then go to new SerialNum byte rom_byte_number and reset mask
      if (rom_byte_mask == 0)
      {
          docrc8(ROM_NO[rom_byte_number]);  // accumulate the CRC
          rom_byte_number++;
          rom_byte_mask = 1;
      }
   }
}
while(rom_byte_number < 8);  // loop until through all ROM bytes 0-7

// if the search was successful then
if (!((id_bit_number < 65) || (crc8 != 0)))
{
   // search successful so set LastDiscrepancy,LastDeviceFlag,search_result
   LastDiscrepancy = last_zero;

   // check for last device
   if (LastDiscrepancy == 0)
      LastDeviceFlag = TRUE;

   search_result = TRUE;
}
```

```
   }

   // if no device found then reset counters so next 'search' will be like a first
   if (!search_result || !ROM_NO[0])
   {
      LastDiscrepancy = 0;
      LastDeviceFlag = FALSE;
      LastFamilyDiscrepancy = 0;
      search_result = FALSE;
   }

   return search_result;
}

//--------------------------------------------------------------------------
// Verify the device with the ROM number in ROM_NO buffer is present.
// Return TRUE  : device verified present
//        FALSE : device not present
//
int OWVerify()
{
   unsigned char rom_backup[8];
   int i,rslt,ld_backup,ldf_backup,lfd_backup;

   // keep a backup copy of the current state
   for (i = 0; i < 8; i++)
      rom_backup[i] = ROM_NO[i];
   ld_backup = LastDiscrepancy;
   ldf_backup = LastDeviceFlag;
   lfd_backup = LastFamilyDiscrepancy;

   // set search to find the same device
   LastDiscrepancy = 64;
   LastDeviceFlag = FALSE;

   if (OWSearch())
   {
      // check if same device found
      rslt = TRUE;
      for (i = 0; i < 8; i++)
      {
         if (rom_backup[i] != ROM_NO[i])
         {
            rslt = FALSE;
            break;
         }
      }
   }
   else
     rslt = FALSE;

   // restore the search state
   for (i = 0; i < 8; i++)
      ROM_NO[i] = rom_backup[i];
   LastDiscrepancy = ld_backup;
   LastDeviceFlag = ldf_backup;
   LastFamilyDiscrepancy = lfd_backup;

   // return the result of the verify
   return rslt;
}

//--------------------------------------------------------------------------
// Setup the search to find the device type 'family_code' on the next call
// to OWNext() if it is present.
//
void OWTargetSetup(unsigned char family_code)
{
   int i;

   // set the search state to find SearchFamily type devices
   ROM_NO[0] = family_code;
   for (i = 1; i < 8; i++)
      ROM_NO[i] = 0;
   LastDiscrepancy = 64;
   LastFamilyDiscrepancy = 0;
   LastDeviceFlag = FALSE;
}

//--------------------------------------------------------------------------
// Setup the search to skip the current device type on the next call
// to OWNext().
//
void OWFamilySkipSetup()
```

```
{
   // set the Last discrepancy to last family discrepancy
   LastDiscrepancy = LastFamilyDiscrepancy;
   LastFamilyDiscrepancy = 0;

   // check for end of list
   if (LastDiscrepancy == 0)
      LastDeviceFlag = TRUE;
}

//--------------------------------------------------------------------------
// 1-Wire Functions to be implemented for a particular platform
//--------------------------------------------------------------------------

//--------------------------------------------------------------------------
// Reset the 1-Wire bus and return the presence of any device
// Return TRUE  : device present
//        FALSE : no device present
//
int OWReset()
{
   // platform specific
   // TMEX API TEST BUILD
   return (TMTouchReset(session_handle) == 1);
}

//--------------------------------------------------------------------------
// Send 8 bits of data to the 1-Wire bus
//
void OWWriteByte(unsigned char byte_value)
{
   // platform specific

   // TMEX API TEST BUILD
   TMTouchByte(session_handle,byte_value);
}

//--------------------------------------------------------------------------
// Send 1 bit of data to teh 1-Wire bus
//
void OWWriteBit(unsigned char bit_value)
{
   // platform specific

   // TMEX API TEST BUILD
   TMTouchBit(session_handle,(short)bit_value);
}

//--------------------------------------------------------------------------
// Read 1 bit of data from the 1-Wire bus
// Return 1 : bit read is 1
//        0 : bit read is 0
//
unsigned char OWReadBit()
{
   // platform specific

   // TMEX API TEST BUILD
   return (unsigned char)TMTouchBit(session_handle,0x01);

}

// TEST BUILD
static unsigned char 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};

//--------------------------------------------------------------------------
// Calculate the CRC8 of the byte value provided with the current
// global 'crc8' value.
```

```
// Returns current global crc8 value
//
unsigned char docrc8(unsigned char value)
{
   // See Application Note 27

   // TEST BUILD
   crc8 = dscrc_table[crc8 ^ value];
   return crc8;
}

//--------------------------------------------------------------------------
// TEST BUILD MAIN
//
int main(short argc, char **argv)
{
   short PortType=5,PortNum=1;
   int rslt,i,cnt;

   // TMEX API SETUP
   // get a session
   session_handle = TMExtendedStartSession(PortNum,PortType,NULL);
   if (session_handle <= 0)
   {
      printf("No session, %d\n",session_handle);
      exit(0);
   }

   // setup the port
   rslt = TMSetup(session_handle);
   if (rslt != 1)
   {
      printf("Fail setup, %d\n",rslt);
      exit(0);
   }
   // END TMEX API SETUP

   // find ALL devices
   printf("\nFIND ALL\n");
   cnt = 0;
   rslt = OWFirst();
   while (rslt)
   {
      // print device found
      for (i = 7; i >= 0; i--)
         printf("%02X", ROM_NO[i]);
      printf("  %d\n",++cnt);

      rslt = OWNext();
   }

   // find only 0x1A
   printf("\nFIND ONLY 0x1A\n");
   cnt = 0;
   OWTargetSetup(0x1A);
   while (OWNext())
   {
      // check for incorrect type
      if (ROM_NO[0] != 0x1A)
         break;

      // print device found
      for (i = 7; i >= 0; i--)
         printf("%02X", ROM_NO[i]);
      printf("  %d\n",++cnt);
   }

   // find all but 0x04, 0x1A, 0x23, and 0x01
   printf("\nFIND ALL EXCEPT 0x10, 0x04, 0x0A, 0x1A, 0x23, 0x01\n");
   cnt = 0;
   rslt = OWFirst();
   while (rslt)
   {
      // check for incorrect type
      if ((ROM_NO[0] == 0x04) || (ROM_NO[0] == 0x1A) ||
          (ROM_NO[0] == 0x01) || (ROM_NO[0] == 0x23) ||
          (ROM_NO[0] == 0x0A) || (ROM_NO[0] == 0x10))
          OWFamilySkipSetup();
      else
      {
         // print device found
         for (i = 7; i >= 0; i--)
            printf("%02X", ROM_NO[i]);
         printf("  %d\n",++cnt);
```

```
      }

      rslt = OWNext();
   }

   // TMEX API CLEANUP
   // release the session
   TMEndSession(session_handle);
   // END TMEX API CLEANUP
}
```

修订历史

01/30/02版1.0—初始版本。

05/16/03版1.1—修改内容：Search ROM命令修改为F0 (十六进制)。

相关型号		
MAX31820	1-Wire环境温度传感器	免费样品
DS1990A	iButton序列号	免费样品
DS1982	iButton 1K位只添加	免费样品
DS2740	高精度库仑计	免费样品
DS2505	16K位只添加存储器	免费样品
DS1973	iButton 4K位EEPROM	免费样品
DS2438	智能电池监测器	免费样品
DS2401	硅序列号	免费样品
DS2408	1-Wire、8通道、可编址开关	免费样品
DS1971	iButton 256位EEPROM	
DS2411	硅序列号，带有V_{CC}输入	免费样品
DS2450	四路、1-Wire A/D转换器	
DS2502	1K位只添加存储器	免费样品
DS1992	iButton 1K位/4K位存储器	免费样品
DS1921G		
DS2762	高精度、Li+电池监测器，带报警输出	免费样品
DS18B20-PAR	1-Wire寄生供电数字温度计	
DS18B20	分辨率可编程设置的1-Wire数字温度计	免费样品
DS1963S	带有SHA-1功能的iButton金融器件	
DS2431	1024位1-Wire EEPROM	免费样品
DS1996	iButton 64K位存储器	免费样品
DS1995	iButton 16K位存储器	免费样品
DS1920	iButton温度记录仪	
DS1985	iButton 16K位只添加	免费样品
DS2411	硅序列号，带有V_{CC}输入	免费样品
DS1904	iButton RTC	
DS1993	iButton 1K位/4K位存储器	免费样品
DS2417	1-Wire时钟芯片，带有中断	免费样品
DS2406	双通道、可编址开关与1K位存储器	免费样品
DS2502	1K位只添加存储器	免费样品

相关型号		
DS2506	64K位只添加存储器	
DS18S20-PAR	寄生供电数字温度计	免费样品
DS18S20	1-Wire寄生供电数字温度计	免费样品
MAX31820PAR		
MAX31826	带有1Kb可锁存EEPROM的1-Wire数字温度传感器	免费样品
DS2432	1Kb、保护型1-Wire EEPROM，带有SHA-1引擎	免费样品
DS2433	4K位1-Wire EEPROM	
DS28E05	1-Wire EEPROM	免费样品

下一步	
EE-Mail	订阅EE-Mail，接收关于您感兴趣的新文档的自动通知。

APP 187: 05 Dec, 2003
应用笔记 187, AN187, AN 187, APP187, Appnote187, Appnote 187