应用笔记 126

# 用软件实现1-WIRE®通信

摘要 : 在没有专用总线主机*(*如*DS2480B*、*DS2482)*的情况下，微处理器可以轻松地产生*1-Wire*时序信号。本应用笔记给出了一个采用*'C'*语言编写、支持标准速率的*1-Wire*主机通信基本子程序实例。*1-Wire*总线的四个基本操作是：复位、写*"1"*、写*"0"*和读数据位。字节操作可以通过反复调用位操作实现，本文提供了通过各种传输线与*1-Wire*器件进行可靠通信的时间参数。

## 引言

在没有专用总线主机的情况下，微处理器可以轻松地产生1-Wire时序信号。本应用笔记给出了一个采用C语言编写、支持标准速率的1-Wire主机通信基本子程序实例。此外，本文也讨论了高速通信模式。要使该实例中的代码正常运行，系统必须满足以下几点要求：

1. 微处理器的通信端口必须是双向的，其输出为漏极开路，且线上具有弱上拉。这也是所有1-Wire总线的基本要求。关于简单的1-Wire主机微处理器电路实例，请参见应用笔记4206："为嵌入式应用选择合适的1-Wire主机"中的1类部分。
2. 微处理器必须能产生标准速度1-Wire通信所需的精确1µs延时和高速通信所需要的0.25µs延时。
3. 通信过程不能被中断。

1-Wire总线有四种基本操作：复位、写1位、写0位和读位操作。在数据资料中，将完成一位传输的时间称为一个时隙。于是字节传输可以通过多次调用位操作来实现，下面的**表1**是各个操作的简要说明以及实现这些操作所必须的步骤列表。**图1**为其时序波形图。**表2**给出了通常线路条件下1-Wire主机与1-Wire器件通信的推荐时间。如果与1-Wire主机相连的器件比较特殊或者线路条件比较特殊，则可以采用最值。请参考可下载的工作表中的系统和器件参数，确定最小值和最大值。

**表1. 1-Wire操作**

| Operation | Description | Implementation |
|---|---|---|
| Write 1 bit | Send a '1' bit to the 1-Wire slaves (Write 1 time slot) | Drive bus low, delay A<br>Release bus, delay B |
| Write 0 bit | send a '0' bit to the 1-Wire slaves (Write 0 time slot) | Drive bus low, delay C<br>Release bus, delay D |
| Read bit | Read a bit from the 1-Wire slaves (Read time slot) | Drive bus low, delay A<br>Release bus, delay E<br>Sample bus to read bit from slave<br>Delay F |
| Reset | Reset the 1-Wire bus slave devices and ready them for a command | Delay G<br>Drive bus low, delay H<br>Release bus, delay I<br>Sample bus, 0 = device(s) present, 1 = no device present<br>Delay J |

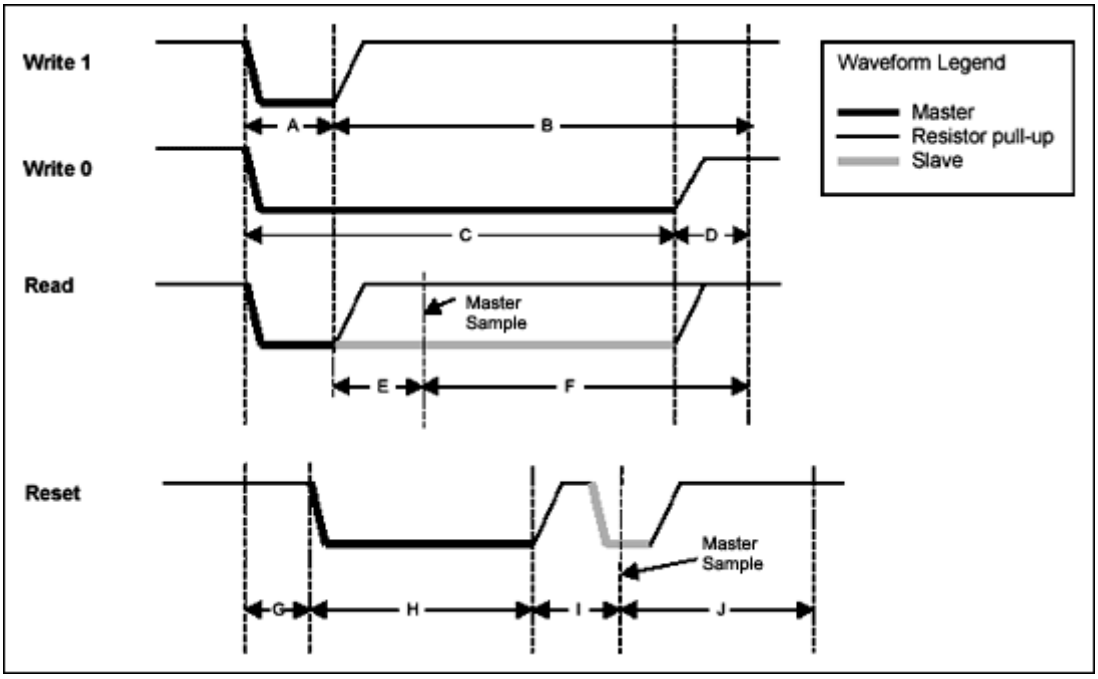

图*1. 1-Wire*时序图

**表2. 1-Wire主机时序**

| Parameter | Speed | Recommended (µs) |
|---|---|---|
| A | Standard | 6 |
| | Overdrive | 1.0 |
| B | Standard | 64 |
| | Overdrive | 7.5 |
| C | Standard | 60 |
| | Overdrive | 7.5 |
| D | Standard | 10 |
| | Overdrive | 2.5 |
| E | Standard | 9 |
| | Overdrive | 1.0 |
| F | Standard | 55 |
| | Overdrive | 7 |
| G | Standard | 0 |
| | Overdrive | 2.5 |
| H | Standard | 480 |
| | Overdrive | 70 |
| I | Standard | 70 |
| | Overdrive | 8.5 |
| J | Standard | 410 |
| | Overdrive | 40 |

计算这些值的工作表可供下载。

## 代码实例

下面代码实例都依赖于两个通用的'C'函数outp和inp，从IO端口读写字节数据。他们通常位于<conio.h>标准库中。当应用于其它平台时，可以采用合适的函数来替代它们。

```
// send 'databyte' to 'port'
int outp(unsigned port, int databyte);

// read byte from 'port'
int inp(unsigned port);
```

代码中的常量PORTADDRESS (图3)用来定义通信端口的地址。这里我们假定使用通信端口的第0位控制1-Wire总线。设定该位为0，将使1-Wire总线变为低电平；设定该位为1，1-Wire总线将被释放，此时1-Wire总线被电阻上拉，或被1-Wire从器件下拉。

代码中的*tickDelay*函数是一个用户编制的子程序，此函数用于产生一个1/4μs整数倍的延时。在不同的平台下，该函数的实现也是不同的，故在此不做具体描述。以下是*tickDelay*函数声明代码，以及一个*SetSpeed*函数，用于设定标准速度和高速模式的延时时间。

实例1. 1-Wire时序的生成

```
// Pause for exactly 'tick' number of ticks = 0.25us
void tickDelay(int tick); // Implementation is platform specific

// 'tick' values
int A,B,C,D,E,F,G,H,I,J;

//--------------------------------------------------------------------------
// Set the 1-Wire timing to 'standard' (standard=1) or 'overdrive' (standard=0).
//
void SetSpeed(int standard)
{
        // Adjust tick values depending on speed
        if (standard)
        {
                // Standard Speed
                A = 6 * 4;
                B = 64 * 4;
                C = 60 * 4;
                D = 10 * 4;
                E = 9 * 4;
                F = 55 * 4;
                G = 0;
                H = 480 * 4;
                I = 70 * 4;
                J = 410 * 4;
        }
        else
        {
                // Overdrive Speed
                A = 1.5 * 4;
                B = 7.5 * 4;
                C = 7.5 * 4;
                D = 2.5 * 4;
                E = 0.75 * 4;
                F = 7 * 4;
                G = 2.5 * 4;
                H = 70 * 4;
                I = 8.5 * 4;
                J = 40 * 4;
        }
}
```

1-Wire基本操作的代码程序如实例2所示。

实例2. 基本的1-Wire函数

```
//-----------------------------------------------------------------------------
// Generate a 1-Wire reset, return 1 if no presence detect was found,
// return 0 otherwise.
// (NOTE: Does not handle alarm presence from DS2404/DS1994)
//
int OWTouchReset(void)
{
        int result;

        tickDelay(G);
        outp(PORTADDRESS,0x00); // Drives DQ low
        tickDelay(H);
        outp(PORTADDRESS,0x01); // Releases the bus
        tickDelay(I);
        result = inp(PORTADDRESS) ^ 0x01; // Sample for presence pulse from slave
        tickDelay(J); // Complete the reset sequence recovery
        return result; // Return sample presence pulse result
}

//-----------------------------------------------------------------------------
// Send a 1-Wire write bit. Provide 10us recovery time.
//
void OWWriteBit(int bit)
{
        if (bit)
        {
                // Write '1' bit
                outp(PORTADDRESS,0x00); // Drives DQ low
                tickDelay(A);
                outp(PORTADDRESS,0x01); // Releases the bus
                tickDelay(B); // Complete the time slot and 10us recovery
        }
        else
        {
                // Write '0' bit
                outp(PORTADDRESS,0x00); // Drives DQ low
                tickDelay(C);
                outp(PORTADDRESS,0x01); // Releases the bus
                tickDelay(D);
        }
}

//-----------------------------------------------------------------------------
// Read a bit from the 1-Wire bus and return it. Provide 10us recovery time.
//
int OWReadBit(void)
{
        int result;

        outp(PORTADDRESS,0x00); // Drives DQ low
        tickDelay(A);
        outp(PORTADDRESS,0x01); // Releases the bus
        tickDelay(E);
        result = inp(PORTADDRESS) & 0x01; // Sample the bit value from the slave
        tickDelay(F); // Complete the time slot and 10us recovery
```

```
        return result;
}
```

该程序包括了1-Wire总线的所有位操作，通过调用该程序可以构成以字节为处理对象的函数，见实例3。

实例3. 派生的1-Wire函数

```
//--------------------------------------------------------------------------
// Write 1-Wire data byte
//
void OWWriteByte(int data)
{
        int loop;

        // Loop to write each bit in the byte, LS-bit first
        for (loop = 0; loop < 8; loop++)
        {
                OWWriteBit(data & 0x01);

                // shift the data byte for the next bit
                data >>= 1;
        }
}

//--------------------------------------------------------------------------
// Read 1-Wire data byte and return it
//
int OWReadByte(void)
{
        int loop, result=0;

        for (loop = 0; loop < 8; loop++)
        {
                // shift the result to get it ready for the next bit
                result >>= 1;

                // if result is one, then set MS bit
                if (OWReadBit())
                        result |= 0x80;
        }
        return result;
}

//--------------------------------------------------------------------------
// Write a 1-Wire data byte and return the sampled result.
//
int OWTouchByte(int data)
{
        int loop, result=0;

        for (loop = 0; loop < 8; loop++)
        {
                // shift the result to get it ready for the next bit
                result >>= 1;

                // If sending a '1' then read a bit else write a '0'
                if (data & 0x01)
                {
                        if (OWReadBit())
                                result |= 0x80;
                }
                else
                        OWWriteBit(0);
```

```
                // shift the data byte for the next bit
                data >>= 1;
        }
        return result;
}

//--------------------------------------------------------------------------
// Write a block 1-Wire data bytes and return the sampled result in the same
// buffer.
//
void OWBlock(unsigned char *data, int data_len)
{
        int loop;

        for (loop = 0; loop < data_len; loop++)
        {
                data[loop] = OWTouchByte(data[loop]);
        }
}

//--------------------------------------------------------------------------
// Set all devices on 1-Wire to overdrive speed. Return '1' if at least one
// overdrive capable device is detected.
//
int OWOverdriveSkip(unsigned char *data, int data_len)
{
        // set the speed to 'standard'
        SetSpeed(1);

        // reset all devices
        if (OWTouchReset()) // Reset the 1-Wire bus
                return 0; // Return if no devices found

        // overdrive skip command
        OWWriteByte(0x3C);

        // set the speed to 'overdrive'
        SetSpeed(0);

        // do a 1-Wire reset in 'overdrive' and return presence result
        return OWTouchReset();
}
```

*OWTouchByte*函数可以同时完成读写1-Wire总线数据，通过该函数可以实现数据块的读写。在一些平台上执行效率更高，Maxim提供的API就采用了这种函数。通过*OWTouchByte*函数，*OWBlock*函数简化了1-Wire总线的数据块发送和接收。注意：*OWTouchByte(0xFF)*与*OWReadByte()*等效，*OWTouchByte(data)*与*OWWriteByte(data)*等效。

这些函数和*tickDelay*函数一起构成了1-Wire总线进行位、字节和块操作时所必需的全部函数。实例4给出了利用这些函数读取DS2432的SHA-1认证页的实例。

实例4. 读DS2432实例

```
//---------------------------------------------------------------------------
// Read and return the page data and SHA-1 message authentication code (MAC)
// from a DS2432.
//
int ReadPageMAC(int page, unsigned char *page_data, unsigned char *mac)
{
        int i;
        unsigned short data_crc16, mac_crc16;

        // set the speed to 'standard'
        SetSpeed(1);

        // select the device
        if (OWTouchReset()) // Reset the 1-Wire bus
                return 0; // Return if no devices found

        OWWriteByte(0xCC); // Send Skip ROM command to select single device

        // read the page
        OWWriteByte(0xA5); // Read Authentication command
        OWWriteByte((page << 5) & 0xFF); // TA1
        OWWriteByte(0); // TA2 (always zero for DS2432)

        // read the page data
        for (i = 0; i < 32; i++)
                page_data[i] = OWReadByte();
        OWWriteByte(0xFF);

        // read the CRC16 of command, address, and data
        data_crc16 = OWReadByte();
        data_crc16 |= (OWReadByte() << 8);

        // delay 2ms for the device MAC computation
        // read the MAC
        for (i = 0; i < 20; i++)
                mac[i] = OWReadByte();

        // read CRC16 of the MAC
        mac_crc16 = OWReadByte();
        mac_crc16 |= (OWReadByte() << 8);

        // check CRC16...
        return 1;
}
```

## 附加软件

本应用笔记给出了1-Wire总线操作的基本函数，这些基本函数都是构建复杂的1-Wire应用的基础。本文忽略的一个重要操作是1-Wire搜索。通过1-Wire搜索可以搜索到挂接在总线上的多个1-Wire从机器件的唯一ID号。在应用笔记187："1-Wire搜索算法"一文中详细介绍了这种搜索方法，同时也给出了实现这些1-Wire基本函数的C程序代码。

1-Wire公共开发包中含有大量针对特定器件编写的代码，由下列链接提供：
http://china.maximintegrated.com/products/ibutton/software/1wire/wirekit.cfm

其它更详细的资料请参阅应用笔记155："1-Wire软件资源指南和器件说明"。

## 替换方案

如果对于某一特定应用，通过软件实现1-Wire主机方案不可行，则作为替换方案，可以采用1-Wire主机芯片或合成的1-Wire主机单元。

Maxim提供了采用Verilog和VHDL编写的1-Wire主机。
DS1WM

如需获取1-Wire主机的Verilog/VHDL代码，请提交技术支持申请。

合成的1-Wire主机工作方式在应用笔记119："在FPGA或ASIC中嵌入1-Wire主机"中进行了说明。

有多种1-Wire主机芯片可以作为微处理器的外设。串行、1-Wire线驱动器DS2480B能够很容易地与标准串行口连接。类似地，DS2482-100、DS2482-101和DS2482-800可以连接至I²C端口。

关于DS2480B的操作，详见应用笔记192："DS2480B串行接口1-Wire线驱动器的使用"。

关于DS2482的操作，详见应用笔记3684："如何使用I²C接口的DS2482 1-Wire主控制器"。

应用笔记244："性能优异的1-Wire网络驱动器"给出了一种专为远距离传输线设计的先进的1-Wire线驱动器。

**修订历史记录**

07/06/00：1.0版—最初版本。
05/28/02：2.0版—纠正了1-Wire复位采样时间。增加了波形图、链接和更多的代码实例。
02/02/04：2.1版—增加了对高速模式的支持，给出了时序的最小值、最大值，更新了代码实例。
09/06/05：2.2版—修正代码例程说明中的PIO极性。
08/04/09：2.3版—增加了AN4206参考内容；修改了高速模式下参数E的值；更正了OWTouchReset示例程序；将1-Wire主机时序中最小值/最大值的计算结果移至工作表中；增加了DS2482参考内容。