Application Note 126
1-Wire Communication Through Software

www.maxim-ic.com

INTRODUCTION

A microprocessor can easily generate 1-Wire® timing signals if a true bus master is not present. This application note provides an example, written in 'C', of the basic standard speed 1-Wire master communication routines. Overdrive communication speed is not covered by this document. There are several system requirements for proper operation of the code examples:

1) The communication port is bidirectional, its output is open-drain, and there is a weak pull-up on the line. This is a requirement of any 1-Wire bus.
2) The system is capable of generating an accurate and repeatable 1µs delay.
3) The communication operations must not be interrupted while being generated.

The four basic operations of a 1-Wire bus are *Reset, Write 1 bit*, *Write 0 bit*, and *Read bit*. The time it takes to perform one bit of communication is called a *time slot* in the device datasheets. Byte functions can then be derived from multiple calls to the bit operations. See Table 1 below for a brief description of each operation and a list of the steps necessary to generate it. Figure 1 illustrates the waveforms graphically. The time values are recommended to produce the most robust 1-Wire master for communication with all 1-Wire devices over various line conditions.

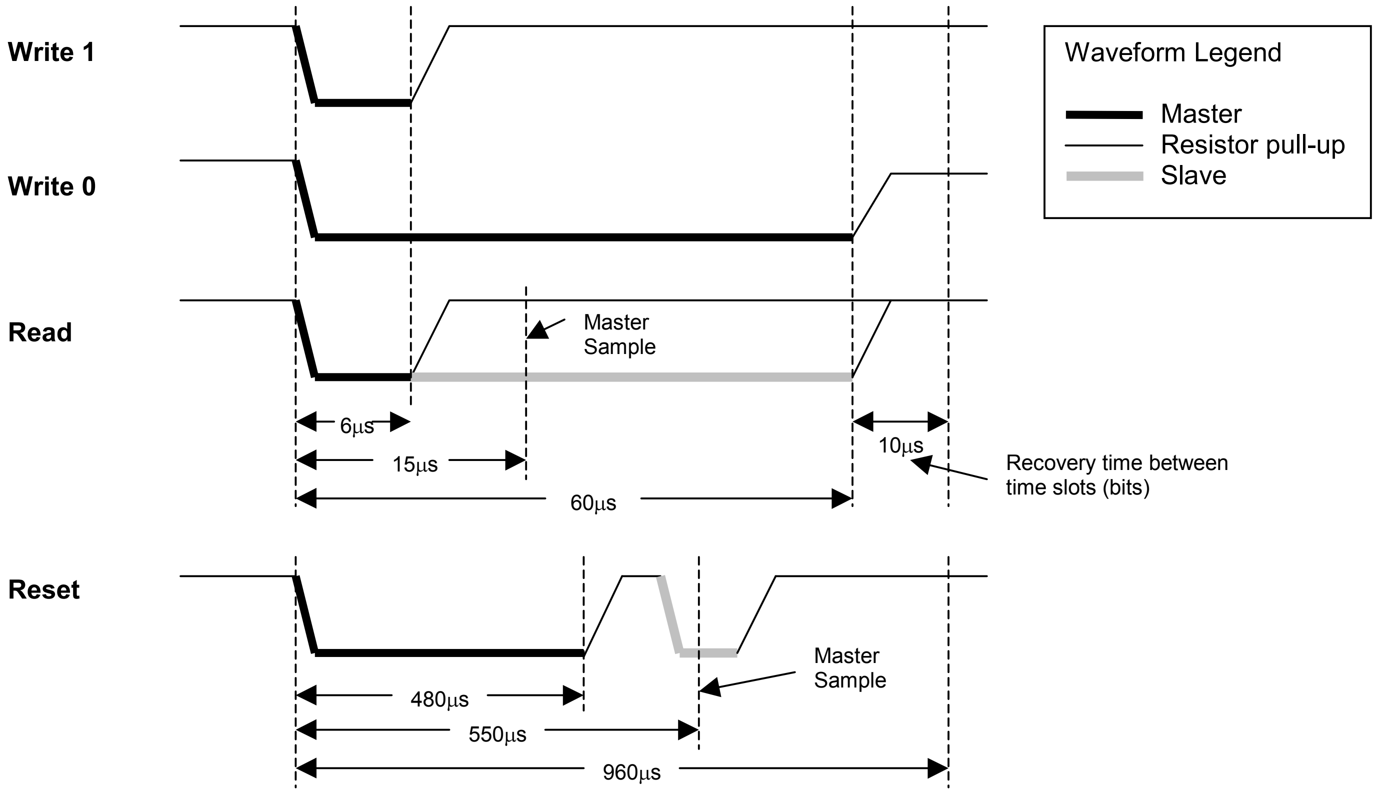
1-WIRE OPERATIONS Table 1

Operation	Description	Implementation
Write 1 bit	Send a '1' bit to the 1-Wire slaves (Write 1 time slot)	Drive bus low, delay 6µs Release bus, delay 64µs
Write 0 bit	send a '0' bit to the 1-Wire slaves (Write 0 time slot)	Drive bus low, delay 60µs Release bus, delay 10µs
Read bit	Read a bit from the 1-Wire slaves (Read time slot)	Drive bus low, delay 6µs Release bus, delay 9µs Sample bus to read bit from slave Delay 55µs
Reset	Reset the 1-Wire bus slave devices and ready them for a command	Drive bus low, delay 480µs Release bus, delay 70µs Sample bus, 0 = device(s) present, 1 = no device present Delay 410µs

Note that the 1-Wire reset operation described here does not take into account the extended (alarming) presence pulse sequence that can be produced by the DS2404 and DS1994 devices. See the device data sheets for this special case. The end of the 1-Wire reset sequence can be sampled to verify that the 1-Wire bus has returned to the pull-up level. If the level is still 0 then the 1-Wire bus may be shorted to ground or an alarming DS2404/DS1994 may be present.

1-Wire is a registered trademark of Dallas Semiconductor.

1-WIRE WAVEFORMS Figure 1



CODE EXAMPLES

This following code samples rely on two common 'C' functions *outp* and *inp* to write and read bytes of data to IO port locations. They are typically located in the <conio.h> standard library. These functions can be replaced by platform appropriate functions.

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

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

The constant PORTADDRESS in the code (see Figure 2) is defined as the location of the communication port. The code assumes bit 0 of this location controls the 1-Wire bus. Setting this bit to 1 will drive the 1-Wire line low. Setting this bit to 0 will release the 1-Wire to be pulled up by the resistor pull-up or pulled-down by a 1-Wire slave device.

The function *usDelay* in the code is a user-generated routine to wait a variable number of microseconds. This function will vary for each unique hardware platform running so it is not implemented here.

```
// Pause for exactly 'microseconds'
void usDelay(int microseconds);
```

Figure 2 below shows the code examples for the primitive 1-Wire operations.

1-WIRE PRIMITIVE FUNCTIONS Figure 2

```
//-----------------------------------------------------------------------------
// 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;

   outp(PORTADDRESS,0x00); // Drives DQ low
   usDelay(480);
   outp(PORTADDRESS,0x01); // Releases the bus
   usDelay(70);

   result = inp(PORTADDRESS) & 0x01;  // Sample for presence pulse from slave

   usDelay(410);           // 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
      usDelay(6);
      outp(PORTADDRESS,0x01); // Releases the bus
      usDelay(64);            // Complete the time slot and 10us recovery
   }
   else
   {
      // Write ‘0’ bit
      outp(PORTADDRESS,0x00); // Drives DQ low
      usDelay(60);
      outp(PORTADDRESS,0x01); // Releases the bus
      usDelay(10);
   }
}

//-----------------------------------------------------------------------------
// 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
   usDelay(6);
   outp(PORTADDRESS,0x01); // Releases the bus
   usDelay(9);

   result = inp(PORTADDRESS) & 0x01; // Sample the bit value from the slave

   usDelay(55);            // Complete the time slot and 10us recovery
   return result;
}
```

This is all for bit-wise manipulation of the 1-Wire bus. The above routines can be built upon to create byte-wise manipulator functions as seen in Figure 3.

DERIVED 1-WIRE FUNCTIONS Figure 3

```
//-----------------------------------------------------------------------------
// 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]);
   }
}
```

The *owTouchByte* operation is a simultaneous write and read from the 1-Wire bus. This function was derived so that a block of both writes and reads could be constructed. This is more efficient on some platforms and is commonly used in API's provided by Dallas Semiconductor. The *OWBlock* function simply sends and receives a block of data to the 1-Wire using the *OWTouchByte* function. Note that *OWTouchByte(0xFF)* is equivalent to *OWReadByte()* and *OWTouchByte(data)* is equivalent to *OWWriteByte(data)*.

These functions plus *usDelay* are all that are required for basic control of the 1-Wire bus at the bit, byte, and block level. The following example in Figure 4 shows how these functions can be used together to read the ICA register of a DS2437.

READ DS2437 EXAMPLE Figure 4

```
//-----------------------------------------------------------------------------
// Read and return the Integrated Current Accumulator
//
int ReadICA(void)
{
   // Recall Page 1
   if (owTouchReset())   // Reset the 1-Wire bus
      return 0;          // Return if no devices found

   owWriteByte(0xCC);    // Send Skip ROM command to select single device
   owWriteByte(0xB8);    // Send Recall memory command
   owWriteByte(0x01);    // Send the Page 1 address

   // Read Page 1
   owTouchReset();       // Reset the 1-Wire bus
   owWriteByte(0xCC);    // Send Skip ROM command to select single device
   owWriteByte(0xBE);    // Send Read memory command
   owWriteByte(0x01);    // Send the Page 1 address
   owReadByte();         // Ignore first 4 bytes in Page 1
   owReadByte();
   owReadByte();
   owReadByte();
   return owReadByte();  // Return the ICA register
}
```

ADDITIONAL SOFTWARE

The basic 1-Wire functions provided in this application note can be used as a foundation to build sophisticated 1-Wire applications. One important operation omitted in this document is the 1-Wire search. The search is a method to discover the unique ID's of multiple 1-Wire slaves connected to the bus. The *1-Wire Search Algorithm* (Application Note 187) describes this method in detail and provides 'C' code that can be used with these primitives.
http://pdfserv.maxim-ic.com/arpdf/AppNotes/app187.pdf

The 1-Wire Public Domain Kit contains a large amount of device-specific code that builds upon what has been provided here.
http://www.ibutton.com/software/1wire/wirekit.html

For details on other resources see the *1-Wire Software Resource Guide* (Application Note 155).
http://pdfserv.maxim-ic.com/arpdf/AppNotes/app155.pdf

ALTERNATIVES

If a software solution is not feasible for a specific application, then a 1-Wire master chip or a synthesized 1-Wire master block can be used as an alternative.

Dallas Semiconductor provides a predefined 1-Wire master in Verilog and VHDL.
http://pdfserv.maxim-ic.com/arpdf/DS1WM.pdf

The 1-Wire master code definition can be downloaded from this link.
ftp://ftp.dalsemi.com/pub/auto_id/licensed/ds1wm.zip

Operation of the synthesizable 1-Wire Master is described in *Communicating through the 1-Wire Master* (Application Note 120) and in *Embedding the 1-Wire Master* (Application Note 119).
http://dbserv.maxim-ic.com/appnotes.cfm?appnote_number=520
http://dbserv.maxim-ic.com/appnotes.cfm?appnote_number=519

There are several 1-Wire master chips that can be used as a peripheral to a microprocessor. The DS2480B Serial 1-Wire Line Driver provides easy connectivity to a standard serial port.
http://pdfserv.maxim-ic.com/arpdf/DS2480B.pdf

Operation of the DS2480B is described in *Using the DS2480B Serial 1-Wire Line Driver* (Application Note 192).
http://pdfserv.maxim-ic.com/arpdf/AppNotes/app192.pdf

The DS1481 provides a 1-Wire master with a parallel interface.
http://pdfserv.maxim-ic.com/arpdf/DS1481.pdf

A more sophisticated 1-Wire line driver designed specifically for long lines is presented in Appendix C of *Guidelines for Reliable 1-Wire Networks* (Application Note 148).
http://pdfserv.maxim-ic.com/arpdf/AppNotes/app148.pdf