

### INTRODUCTION

A microprocessor can easily generate 1-Wire<sup>®</sup> timing signals if a dedicated 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 also covered by this document. There are several system requirements for proper operation of the code examples:

- 1) The communication port must be 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. See Appendix A in *Guidelines for Reliable 1-Wire Networks* (Application Note 148) for a simple example of a 1-Wire master microprocessor circuit.
- 2) The system must be capable of generating an accurate and repeatable 1μs delay for standard speed and 0.25μs delay for overdrive speed.
- 3) The communication operations must not be interrupted while being generated.

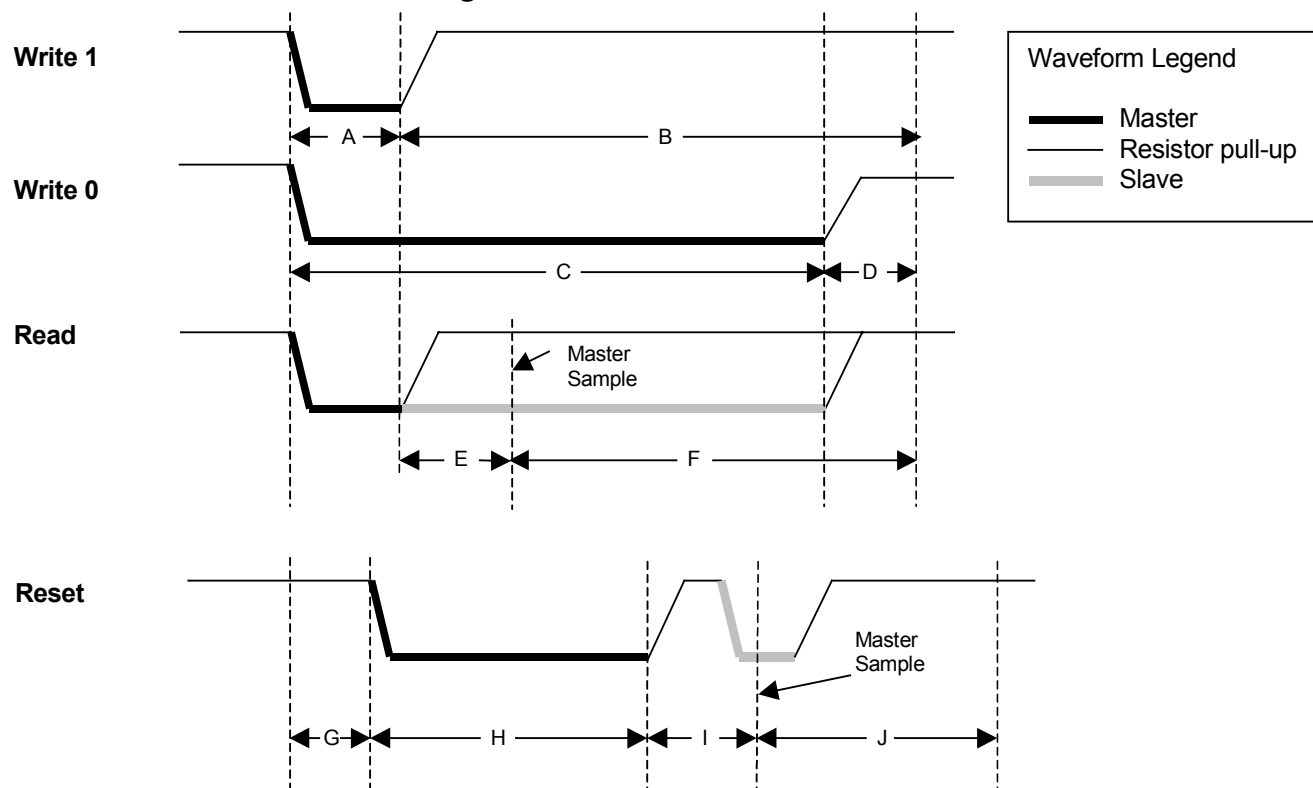
The four basic operations of a 1-Wire bus are *Reset*, *Write 1 bit*, *Write 0 bit*, and *Read bit*. The time it takes to perform one bit of communication is called a *time slot* in the device datasheets. Byte functions can then be derived from multiple calls to the bit operations. See Table 1 below for a brief description of each operation and a list of the steps necessary to generate it. Figure 1 illustrates the waveforms graphically. Table 2 shows the minimum, maximum and recommended timings for the 1-Wire master to communicate with 1-Wire devices over the most common line conditions. Alternate minimum and maximum values can be used when restricting the 1-Wire master to a particular set of devices and 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 A 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 Release bus, delay D
Read bit	Read a bit from the 1-Wire slaves (Read time slot)	Drive bus low, delay A Release bus, delay E Sample bus to read bit from slave Delay F
Reset	Reset the 1-Wire bus slave devices and ready them for a command	Delay G Drive bus low, delay H Release bus, delay I Sample bus, 0 = device(s) present, 1 = no device present Delay J

1-Wire is a registered trademark of Dallas Semiconductor.

## 1-WIRE WAVEFORMS Figure 1



## 1-WIRE MASTER TIMING Table 2

Parameter	Speed	Min ( $\mu\text{s}$ )	Recommended ( $\mu\text{s}$ )	Max ( $\mu\text{s}$ )	Notes
A	Standard	5	6	15	1, 2
	Overdrive	1	1.5	1.85	1, 3
B	Standard	59	64	N/A	2, 4
	Overdrive	7.5	7.5	N/A	3, 4
C	Standard	60	60	120	2, 5
	Overdrive	7	7.5	14	3, 5
D	Standard	8	10	N/A	2, 6
	Overdrive	2.5	2.5	N/A	3, 6
E	Standard	5	9	12	2, 7, 8
	Overdrive	0.5	0.75	0.85	3, 7, 8
F	Standard	50	55	N/A	2, 9
	Overdrive	6.75	7	N/A	3, 9
G	Standard	0	0	0	
	Overdrive	2.5	2.5	N/A	3, 14
H	Standard	480	480	640	2, 10, 15
	Overdrive	68	70	80	3, 10
I	Standard	63	70	78	2, 11
	Overdrive	7.2	8.5	8.8	3, 11
J	Standard	410	410	N/A	2, 12, 13
	Overdrive	39.5	40	N/A	3, 12

Worksheet to calculate these values can be found at: [ftp://ftp.dalsemi.com/pub/auto\\_id/public/an126.zip](ftp://ftp.dalsemi.com/pub/auto_id/public/an126.zip)

**Notes**

1. Expressed as  $t_{WIL}$  (write 1 low) minus  $\epsilon$  (rise time to  $V_{TH}$ ) plus  $t_F$  (fall time to  $V_{TL}$ ) in the device datasheets.
2. Assume a mid-range standard speed network with a rise and fall time of less than  $3\mu s$ .
3. Assume a small overdrive speed network with a rise and fall time less than  $0.5\mu s$ .
4. Expressed as  $t_{SLOT}$  (time slot length) in the device datasheets minus the value used for 'A'.
5. Expressed as  $t_{WOL}$  (write 0 low) minus  $\delta$  (rise time to  $V_{IHMASTER}$ ) plus  $t_F$  (fall time to  $V_{TL}$ ) in the device datasheets.
6. Expressed as  $t_{REC}$  (recovery time) plus  $\delta$  (rise time to  $V_{IHMASTER}$ ) in the device datasheets.
7. Expressed as  $t_{MSR}$  (master sample read) plus  $t_F$  (fall time to  $V_{TL}$ ) in the device datasheets minus the value used for 'A'.
8. Sample as late as possible in the range to provide the most rise time recovery.
9. Expressed as  $t_{SLOT}$  (time slot length) in the device datasheets minus the value used for 'A' minus the value used for 'E'.
10. Expressed as  $t_{RSTL}$  (reset low time) minus  $\epsilon$  (rise time to  $V_{TH}$ ) plus  $t_F$  (fall time to  $V_{TL}$ ) in the device datasheets.
11. Expressed as  $t_{MSP}$  (master sample presence) plus  $\epsilon$  (rise time to  $V_{TH}$ ) in the device datasheets.
12. Min expressed as  $t_{RSTL}$  (reset low time) in the device datasheets minus the value used for 'I'.
13. 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.
14. Expressed as  $t_{REC}$  (recovery time) minus the value used for 'D'. Some devices in overdrive require an extra delay before a  $t_{RSTL}$  (reset low time) to make sure that the device's parasite power supply is fully charged.
15. For low-voltage operation, some devices need a longer value. See device datasheet for the applicable value.

**CODE EXAMPLES**

The 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 3) 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 *tickDelay* in the code is a user-generated routine to wait a variable number of  $\frac{1}{4}$  microseconds. This function will vary for each unique hardware platform running so it is not implemented here. Below is the function declaration for the *tickDelay* along with a function *SetSpeed* to set the recommended standard and overdrive speed tick values.

## 1-WIRE TIMING GENERATION Figure 2

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

Figure 3 below shows the code examples for the basic 1-Wire operations.

## 1-WIRE BASIC FUNCTIONS Figure 3

```

//-----
// 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;
}

```

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 4.

**DERIVED 1-WIRE FUNCTIONS Figure 4**

```

//-----
// 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();
}

```

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 *tickDelay* 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 5 shows how these functions can be used together to read a SHA-1 authenticated page of the DS2432.

## READ DS2432 EXAMPLE Figure 5

```

//-----
// 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 0 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;
}

```

## 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 basic 1-Wire functions.

<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](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=520)

[http://dbserv.maxim-ic.com/appnotes.cfm?appnote\\_number=519](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 DS2490 provides a 1-Wire master with a USB interface.

<http://pdfserv.maxim-ic.com/en/ds/DS2490.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 *Advanced 1-Wire Network Driver* (Application Note 244).

<http://pdfserv.maxim-ic.com/en/an/app244.pdf>

### Revision History

07/06/00: Version 1.0 – Initial release.

05/28/02: Version 2.0 – Correct 1-Wire reset sample time. Add wave figure, links, and more code examples.

02/02/04: Version 2.1 – Add overdrive support, provided min/max on timings, and update example.