

APPLICATION NOTE 120

Using an API to Control the DS1WM 1-Wire® Bus Master

Abstract: This application note presents examples of C code that utilizes a Maxim API to control the DS1WM 1-Wire Bus Master. The discussion assumes that the DS1WM has been designed into an FPGA or ASIC and that a host CPU controls the DS1WM through API calls. The DS1WM communicates through two files (DS1WM.c and DS1WM.h) which makeup the API. Topics of discussion include examples of initialization, 1-Wire reset, and the Search ROM algorithm. Examples illustrate how to utilize common 1-Wire features. The reader should be knowledgeable about 1-Wire devices, the DS1WM 1-Wire Master, and the 1-Wire protocol.

Introduction

The [DS1WM](#) 1-Wire bus master eliminates CPU bit-banging by internally generating 1-Wire timing and control signals. This control function lets the system programmer focus on program development using the API functions provided. The DS1WM API is written in ANSI C, which makes it portable to many microprocessor platforms supporting ANSI C. The following examples demonstrate how to identify, select, and communicate with 1-Wire slave devices on the network.

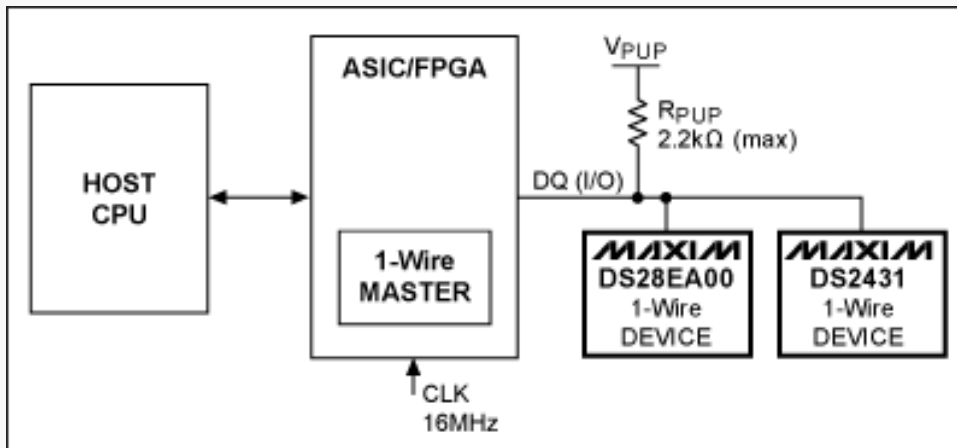


Figure 1. Example circuit of a 1-Wire network bus.

The circuit configuration in **Figure 1** is used in all the following examples. The host CPU uses the DS1WM to communicate with a [DS28EA00](#) 1-Wire Digital Temperature Sensor with chain-mode and GPIO, and with the [DS2431](#) 1-Wire 1Kb EEPROM. The examples showcase the API, not the slave devices' functionality. A 16MHz system clock drives the 1-Wire master's timing through the CLK pin. The 1-Wire master's port pins need to be mapped to the applicable microprocessor pins. The pin maps can be found in the API file, DS1WM.h. The MaxNumberDevices macro found in the API header file needs to be changed to the maximum number of devices that could potentially be found. The ReadByteFromRegister and WriteByte2Register functions found in the DS1WM.c API file will also need to be modified to appropriately toggle the microprocessor port pins. The V_{PUP} voltage is typically at 5.0V with an R_{PUP} no higher than 2.2kΩ.

Discussion Overview

The DS1WM and the API are provided free of charge when requested through [Maxim Support](#). Initialization and 1-Wire reset control for DS1WM code are discussed below. An example shows how to retrieve the unique registration numbers (ROM ID) for each device by using the DS1WM's Search ROM Accelerator. Using the stored registration numbers, each 1-Wire slave device can be identified by its Family Code and accessed using the Match ROM function. See application note 155, "[1-Wire® Software Resource Guide Device Description](#)," for a table of family codes. The DS28EA00 example demonstrates how to perform a temperature conversion. The second example shows two methods for writing and reading the scratchpad memory of a DS2431 in overdrive speed.

Basic Bus Master Operation

The microprocessor initializes the DS1WM by toggling the reset pin. The API interface performs all 1-Wire communications. The example operations are described by calling the functions of the DS1WM API. Due to microprocessor code variations for implementing interrupts, the subsequent examples use polling instead of interrupt-driven communications.

The 1-Wire operations in **Table 1** are used in the following code examples.

Table 1. 1-Wire Operations

Operation*	Description
OWReset	Sends the 1-Wire reset stimulus and checks for the pulses of 1-Wire slave devices that are present on the bus.
OWWriteByte/OWReadByte	Sends or receives a single byte of data from the 1-Wire bus.
OWWriteBytes/OWReadBytes	Sends or receives multiple bytes of data from the 1-Wire bus.
OWSearch	Performs the 1-Wire Search Algorithm (see application note 187, " 1-Wire Search Algorithm ").
OverdriveEnable	Sets the 1-Wire communication speed for the DS1WM to overdrive. Note that this only changes the communication speed of the DS1WM; the 1-Wire slave device must be instructed to make the switch when going from normal to overdrive. The 1-Wire slave will always revert to standard speed when it encounters a standard-speed 1-Wire reset.
OverdriveDisable	Sets the 1-Wire communication speed for the DS1WM to standard. Note that this only changes the communication speed of the DS1WM; a standard-speed 1-Wire reset is required for slave devices to exit overdrive.
MatchROM	Selects device by issuing the Match ROM command followed by the 64-bit ROM ID selected.
SetClockFrequency	Sets the clock frequency for the DS1WM.
InterruptEnableRegisterWrite	Writes a single byte of data to the DS1WM Interrupt Enable register.
InterruptRegisterRead	Reads a single byte of data from the DS1WM Interrupt register.
ReceiveBufferRead	Reads a single byte of data from the DS1WM Receive Buffer register.

*This table covers routines used in the examples. Please refer to the DS1WM.c file for all descriptions.

Initialization

The host begins by initializing the DS1WM for proper 1-Wire bus timing. The host passes the value 16 (which represents frequency) to the SetClockFrequency routine, which writes a 0x90h to the Clock Divisor register. (See the [DS1WM data sheet](#).) The Interrupt Enable register is set to 0x00h to prevent interrupts from being generated on the INTR pin. On power-up, the receive buffer may have invalid values, so it is a good practice to flush the receive buffer before sending any 1-Wire commands. The Interrupt register and the Receive Buffer are read once at startup to clear any bits.

```
//-----  
//Start of initialization example  
  
SetClockFrequency(16);           //Set clock frequency to 16MHz (power-on default)  
InterruptEnableRegisterWrite(0x00); //Clear interrupts  
  
//Flush receive buffer  
InterruptRegisterData = InterruptRegisterRead();  
ReceiveBufferRead();  
  
//End of initialization example  
//-----
```

1-Wire Reset (OWReset API Function)

After initialization, the host must determine whether there are any devices on the 1-Wire bus. To do this, the host calls the OWReset function. This function returns a 1 if a presence detect pulse was sensed, or a 0 if no device was detected or an error occurred. If a 0 is returned, the ErrorStatus variable should be checked for the error condition. A 1-Wire reset must be issued prior to any 1-Wire command (i.e., Match ROM, Skip ROM, Read ROM), except when calling the OWSearch function, which contains its own reset. The software designer should add appropriate error-handling code to be triggered by the error condition. The ErrorStatus value can mean that one of the following errors has happened: no presence detect occurred; no

part was found; there was a 1-Wire short; or the 1-Wire is stuck low.

```
Result = OWReset();
if(!Result){
    switch(ErrorStatus){
        case -1:          //DS1WM did not recognize 1-Wire reset (PD=0)
            //To do: add your error code here
            break;
        case -2:          //No device found (PDR=1)
            //To do: add your error code here
            break;
        case -7:          //1-Wire IO is shorted (OW_SHORT=1)
            //To do: add your error code here
            break;
        case -8:          //1-Wire IO is shorted (OW_LOW=1)
            //To do: add your error code here
            break;
    }
}
```

Finding ROM IDs with the Search ROM Accelerator

Each 1-Wire device on the bus needs to be identified. A unique 64-bit ROM ID, stored in every 1-Wire device, is used to select individual devices; it identifies the type of device based on the Family Code. To reduce the complexity of the example code, the 8-bit family code will be used to distinguish between the DS28EA00 and the DS2431.

Note: If more than one device type (i.e., with the same Family Code) is attached to the 1-Wire line, then the following methods could not distinguish one device from the other. Without some way of distinguishing between multiple 1-Wire devices on a bus, there is no way to communicate function commands to a particular 1-Wire device. If two or more similar devices are used, a lookup table must be implemented in order to appropriately access the intended device.

When called, the OWSearch function identifies all the 1-Wire devices on the bus; it populates the 8-byte ROM ID into the array passed. There is no need to place a call to the OWReset function since it is already part of the OWSearch function. The OWSearch function has this exception due to the iterative process of the search algorithm. Specifically, the OWSearch functions returns the number of 1-Wire devices found, and places each of the 64-bit ROM IDs into a two-dimensional array called ROMCodes.

The examples below only show the OWSearch called once, therefore, any devices subsequently added to the 1-Wire bus will not be recognized by this code. When successful, the OWSearch will return the number of devices found. If the search was not successful, the ErrorStatus variable will return the value of the error, which includes error conditions for the OWReset. After a successful search, the ROMCodes array is traversed and the array index for each device is saved. This device indexing is done by comparing the Family Code for each device found. The array index can then be called later to communicate to the particular 1-Wire device that was found. Essentially, this creates a simple linked list. (Note: This could be implemented with pointers.)

```
//-----
//Start of DS1WM search ROM accelerator example

//Devices on the 1-Wire IO are:
//DS28EA00      Family Code = 42h (6900000004E8C842)
//DS2431        Family Code = 2Dh (5A0000000FDE052D)

//Find all devices on 1-Wire line and populate ROMCodes array
Result = OWSearch(ROMCodes);    //Returns number of devices found if successful

//Set number of 1-Wire devices found
if(Result)
    NumberOfDevices = Result;

if(!Result){
    switch(ErrorStatus){
        case -1:          //DS1WM did not recognize 1-Wire Reset (PD=0)
            //To do: add your error code here
            break;
```

```

    case -2:          //No device found (PDR=1)
        //To do: add your error code here
        break;
    case -7:          //1-Wire IO is shorted (OW_SHORT=1)
        //To do: add your error code here
        break;
    case -8:          //1-Wire IO is shorted (OW_LOW=1)
        //To do: add your error code here
        break;
    case -9:          //Invalid CRC for device
        //To do: add your error code here
        break;
    case -10:         //ROMCodes array too small (Edit MaxNumberDevices in DS1WM.h)
        //To do: add your error code here
        break;
}
}

//Note: This function is intended to be used when there is only one device with the same
//Family Code present on the line
for(i=0;i<NumberOfDevices;i++){
    if(ROMCodes[i][0] == 0x42){
        DS28EA00 = i;          //Save off array index for DS28EA00
        continue;
    }

    if(ROMCodes[i][0] == 0x2D){
        DS2431 = i;          //Save off array index for DS2431
        continue;
    }
}
}

```

Using DS1WM API Functions to Control the DS28EA00

This example uses the ROM ID found earlier to communicate to the DS28EA00. An OWReset followed by a Match ROM command is issued. The ROM Code array, along with the index of the device to access, is passed to the Match ROM function. Devices on the 1-Wire bus will see the Match ROM command 0x55h followed by a 64-bit ROM sequence that allows the bus master to address the specific DS28EA00 on a multidrop bus. Only the DS28EA00 that exactly matches the 64-bit ROM sequence responds to the next set of commands; all other slave devices wait for a reset pulse.

The OWWriteByte and OWReadByte API functions are used to issue memory commands (i.e., Write/Copy/Read Scratchpad commands). The DS1WM sets up the temperature alarms and resolution by first sending a Write Scratchpad command (0x4Eh) followed by the values to program the Temperature High, Temperature Low, and Configuration registers. After completion, another OWReset is issued with a Match ROM command, ROM ID, and Copy Scratchpad command (0x48h) that copies the scratchpad contents into register memory to complete the temperature resolution setup. A proper delay of 10ms must be added for your CPU host to allow time for the copy. Due to variations in microprocessor delay routines, the API includes only commented pseudo code.

Following the copy, an OWReset is again issued followed by the Match ROM, ROM ID, and the Convert Temperature command (0x44h). A delay of 100ms is required to allow time for the temperature conversion to occur. Finally, an OWReset is issued, followed by the Match ROM, ROM ID, and the Read Scratchpad command (0xBEh) to read the two bytes containing the temperature data. Notice the pattern of always selecting the DS28EA00 with an OWReset followed by a Match ROM command and ROM ID whenever the device needs to be selected. The Skip ROM command should be used only if there is a single device on the line (i.e., no Search ROM required).

```

//-----
//Start of DS28EA00 example

Result = OWReset();
if(!Result){
    switch(ErrorStatus){
        case -1:          //DS1WM did not recognize 1-Wire reset(PD=0)
            //To do: add your error code here
            break;
        case -2:          //No device found(PDR=1)
            //To do: add your error code here
    }
}

```

```

        break;
    case -7:          //1-Wire IO is shorted(OW_SHORT=1)
        //To do: add your error code here
        break;
    case -8:          //1-Wire IO is shorted(OW_LOW=1)
        //To do: add your error code here
        break;
    }
}

//Set temperature resolution
Result = MatchROM(ROMCodes,DS28EA00);          //Select device
Result = OWWriteByte(0x4E);                    //Issue Write Scratchpad command
Result = OWWriteByte(0x00);                    //TH register data
Result = OWWriteByte(0x00);                    //TL degister data
Result = OWWriteByte(0x1F);                    //Config. reg. data (set 9-bit temp. resolution)

OWReset();                                     //Error code removed for conciseness
MatchROM(ROMCodes,DS28EA00);                   //Select device
OWWriteByte(0x48);                              //Issue Copy Scratchpad command

//To do: add microprocessor-specific code delay to allow copy to complete
//Delay(10MS);

//Psuedo code

OWReset();                                     //1-Wire reset
MatchROM(ROMCodes,DS28EA00);                   //Select device
OWWriteByte(0x44);                              //Issue Convert Temperature command

//To do: add microprocessor-specific code delay to allow temperature conversion to complete
//Delay(100MS);

//Psuedo code

//Read temperature results from scratchpad
OWReset();                                     //1-Wire reset
MatchROM(ROMCodes,DS28EA00);                   //Select device
OWWriteByte(0xBE);                              //Issue Read Scratchpad command
TempLSB = OWReadByte();                         //Read byte
TempMSB = OWReadByte();                         //Read byte
//End of DS28EA00 example
//-----

```

Using DS1WM API Functions to Control the DS2431 in Overdrive

This example uses the ROM ID found earlier to communicate with the DS2431. An OWReset is issued, followed by an Overdrive Skip ROM command (0x3Ch), which places all overdrive-supporting 1-Wire devices into overdrive. The OverdriveEnable function is called to activate the DS1WM overdrive timing. All 1-Wire functions now operate at overdrive speed. Application note 126, "[1-Wire® Communication Through Software](#)," details 1-Wire timing in both standard and overdrive modes.

Using Target Address TA1 and TA2 Variables

In the first example method, the user variables TA1 and TA2 are set to the desired memory address of the DS2431. An OWReset is issued, followed by a Match ROM, DS2431 ROM ID, Write Scratchpad command (0x0Fh), Target Address 1 & 2, and 8 bytes of data written into the 64-bit DS2431 scratchpad. A read back and check of the CRC16 is recommended. Application note 27, "[Understanding and Using Cyclic Redundancy Checks with Maxim iButton® Products](#)," discusses the CRC16 in greater detail.

OWWriteBytes and OWReadBytes

In the second example method, two new API functions are used called: OWWriteBytes and OWReadBytes. These two API functions simplify writing and reading large amounts of data to the scratchpad.

The write scratchpad method is: set Target address 1 & 2; write data to WriteBytes array; issue an OWReset; send Match ROM, DS2431 ROM ID, Write Scratchpad command (0x0Fh); send Target Address 1 & 2; write all 10 bytes stored in WriteBytes using OWWriteBytes function; read back CRC16 and check CRC16 if desired.

The read scratchpad method is: issue an OWReset; send Match ROM, DS2431 ROM ID; issue Read Scratchpad command (0xAAh); read all 13 bytes (TA1, TA2, ES, CRC16 & 8 bytes of data) using OWReadBytes function; and store in ReadBytes.

To exit the overdrive mode, call the API function OverdriveDisable followed by a standard OWReset, which returns all devices to standard speed.

```
//-----  
//Start of DS2431 example  
  
Result = OWReset(); //Error code removed for conciseness  
Result = OWWriteByte(0x3C); //Overdrive Skip ROM (all devices are now in overdrive)  
OverdriveEnable(); //Enable Overdrive Mode  
  
//Write scratchpad with data  
//First method  
TA1 = 0x00;  
TA2 = 0x00;  
Result = OWReset(); //1-Wire reset  
Result = MatchROM(ROMCodes,DS2431); //Select device  
Result = OWWriteByte(0x0F); //Issue Write Scratchpad command  
Result = OWWriteByte(TA1); //Send target address 1 (TA1)  
Result = OWWriteByte(TA2); //Send target address 2 (TA2)  
  
//Write 8 Bytes of Data  
Result = OWWriteByte(0x11); //Send Data Byte for all  
Result = OWWriteByte(0x22);  
Result = OWWriteByte(0x33);  
Result = OWWriteByte(0x44);  
Result = OWWriteByte(0x55);  
Result = OWWriteByte(0x66);  
Result = OWWriteByte(0x77);  
Result = OWWriteByte(0x88);  
  
//It is recommended that the CRC16 be read back and verified  
//CRC16 code was left out for conciseness  
  
Result = OWReset(); //1-Wire Reset  
Result = MatchROM(ROMCodes,DS2431); //Select device  
Result = OWWriteByte(0xAA); //Issue Read Scratchpad command  
Result = OWReadByte(); //Read TA1  
if(Result != TA1){  
    //To do: Add your error code here  
}  
Result = OWReadByte(); //Read TA2  
if(Result != TA2){  
    //To do: Add your error code here  
}  
  
ES = OWReadByte(); //Read ES  
  
//To do: add your error code after reads  
Result = OWReadByte(); //Read Data Byte (0x11)  
Result = OWReadByte(); //Read Data Byte (0x22)  
Result = OWReadByte(); //Read Data Byte (0x33)  
Result = OWReadByte(); //Read Data Byte (0x44)  
Result = OWReadByte(); //Read Data Byte (0x55)  
Result = OWReadByte(); //Read Data Byte (0x66)  
Result = OWReadByte(); //Read Data Byte (0x77)  
Result = OWReadByte(); //Read Data Byte (0x88)  
  
//It is recommended that the CRC16 be read back and verified  
//CRC16 code was left out for conciseness
```

```

//Second method
TA1 = 0x00;
TA2 = 0x00;

WriteBytes[0] = TA1;
WriteBytes[1] = TA2;

for(i=2;i<10;i++){
    WriteBytes[i] = i;
}

Result = OWReset(); //1-Wire reset
Result = MatchROM(ROMCodes,DS2431); //Select device
Result = OWWriteByte(0x0F); //Issue Write Scratchpad command

//Write 10 bytes of data (TA1, TA2 & 8 bytes of data)
OWWriteBytes(WriteBytes,10); //Write data bytes

//It is recommended that the CRC16 be read back and verified
//CRC16 code was left out for conciseness

Result = OWReset(); //1-Wire reset
Result = MatchROM(ROMCodes,DS2431); //Select device
Result = OWWriteByte(0xAA); //Issue Read Scratchpad command

//Read 13 bytes of data (TA1, TA2, ES, CRC16 & 8 bytes of data)
OWReadBytes(ReadBytes,13); //Read data bytes

//It is recommended that the CRC16 be read back and verified
//CRC16 code was left out for conciseness

//Exit overdrive
OverdriveDisable();
Result = OWReset(); //Std. reset issued (all devices are now in standard speed)

//End of DS2431 example
//-----

```

Conclusion

This application note provides examples for using the API functions to control the DS1WM without the overhead of having the host CPU generate 1-Wire timing. One should now be familiar with the generic API functions to select and access multiple 1-Wire devices on the bus. Control and access of the DS28EA00 and the DS2431 were used as example devices on the bus. Also overdrive and single/multibyte reading and writing were shown.

1-Wire is a registered trademark of Maxim Integrated Products, Inc.
 iButton is a registered trademark of Maxim Integrated Products, Inc.

Application note 120: www.maxim-ic.com/an120

More Information

For technical support: www.maxim-ic.com/support

For samples: www.maxim-ic.com/samples

Other questions and comments: www.maxim-ic.com/contact

Automatic Updates

Would you like to be automatically notified when new application notes are published in your areas of interest? [Sign up for EE-Mail™](#).

Related Parts

DS18B20: [QuickView](#) -- [Full \(PDF\) Data Sheet](#) -- [Free Samples](#)

DS1WM: [QuickView](#) -- [Full \(PDF\) Data Sheet](#)

DS2408: [QuickView](#) -- [Full \(PDF\) Data Sheet](#) -- [Free Samples](#)

DS2502-E48: [QuickView](#) -- [Full \(PDF\) Data Sheet](#) -- [Free Samples](#)

AN120, AN 120, APP120, Appnote120, Appnote 120

Copyright © by Maxim Integrated Products

Additional legal notices: www.maxim-ic.com/legal