

FROM THE BENCH

Jeff Bachiochi

Dallas 1-Wire

Part 1: One on One



Hey, it's not just one time only! Jeff opens

this new series on the Dallas 1-wire bus protocol by discussing 1-wire devices and how they are accessed when you use them alone on a single bidirectional I/O pin.



Those of you who work with small embedded micros know that there are always tradeoffs to be made between cost and function. You pay more, you get more. Whether it's code space, internal RAM, peripherals, or I/O, the more you need the more it costs.

It's no wonder that when designs change (via managerial enhancements), designers often go nuts trying to cram in the necessary bells and whistles. Sure, great designs maximize the use of available assets. Just don't expect tomorrow's bright idea to be easily implemented with today's minimalist designs. Maybe we engineers have to start outsmarting management by overcompensating for the anticipated deluge of last minute must-haves.

There are a number of successful approaches to adding function through external devices without demanding more and more I/O from a processor. To keep the required I/O to a minimum, these approaches tend to use some kind of serial protocol. SPI and I²C are two of the most popular.

SPI uses up to four signal lines (SCL,

SDI, SDO, and CS), whereas I²C uses two signal lines (CLK and DATA). SPI is a shorter protocol because each device has its own chip select. I²C requires address information to be sent along with the data so it's a longer protocol. But, it only requires two signal lines, even when multiple devices share the same I/O.

THE CHALLENGE

Is it possible to reduce the necessary interface to a single I/O pin, yet

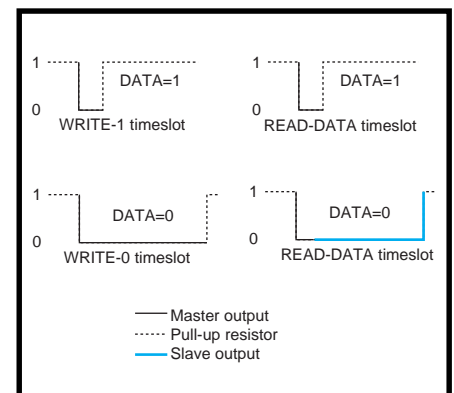


Figure 1—After an initial low output, both write and read timeslots can hold the output low indicating a data bit of 0. When writing, the master controls the data, and when reading, the slave controls the data.

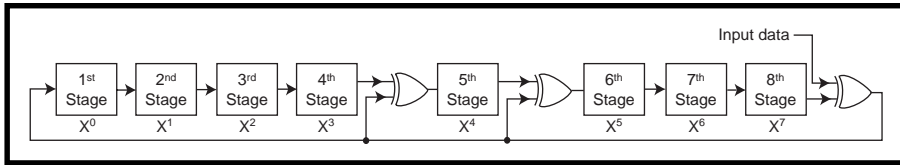


Figure 2—A cyclic redundancy check (CRC) provides some assurance that the received data is good. The last byte of a transmission holds the CRC value calculated from the previous data bytes.

still let multiple external devices communicate with a processor? Dallas Semiconductor accepted the challenge years ago by creating a line of 1-wire data devices.

To fit a 1-wire protocol, the device needs to communicate using a half-duplex protocol. This arrangement achieves bidirectional communication over the same wire.

The Dallas design is based on an open-collector type drive with a pull-up resistor to V_{CC} . Any device connected to the 1-wire bus can pull the idle state (bus held high by the pullup) down to ground with its open-collector output.

The computer or microcomputer that the external 1-wire device is connected to is considered the master. All external devices are considered slaves. Generally, the slave devices won't clamp the bus low unless they are responding to the master's request.

Most 1-wire devices are powered parasitically, which means that they derive their power from the bus (while it's pulled up). This setup gives their timed circuits energy to respond to a falling edge on the bus.

Communication between the master and slave devices is handled via read and write timeslots. A timeslot is a predetermined period of time that begins when an active state (low) is applied to the 1-wire bus. The falling edge of this short pulse lets the other devices know when to read the data bus or write to the data bus for the remainder of the time slot.

There are two different slot timings—reset and data. The reset slot contains a low for at least 480 μ s and a response time of not more than 300 μ s. These long times are easily recognizable from the shorter data-slot times.

Data slots are no more than 120 μ s. These consist of a maximum 15 μ s low followed by the actual data, which is a high or low on the bus. The master's

initial low gets all external devices listening to the bus.

If the external device is receiving data, it samples the bus after a minimum of 15 μ s from the initial drop of the bus. If the external device must send data, the data is placed on the bus immediately following the initial drop (see Figure 1).

Remember that a slave device won't initiate data transfer on the bus. There must be a way for the master to tell an external device to respond. When a single device is part of the system, either as a permanent component or a socketed temporary touch device, a reset pulse should precede any commands.

Although devices that are permanently attached will always be present, temporary devices may or may not be there. The reset pulse is a request for any connected devices to respond with a presence pulse to indicate that at least one device is connected.

ONE ON ONE

Listing 1—With these assembly-language routines, you can use a bidirectional bit to talk to any Dallas 1-wire device.

```

ASM
_RESET_P  bcf     PORTB,0    ;Force 1-wire I/O bit low
           bsf     STATUS,5  ;Point to Bank1 (direction control)
           bcf     TRISB,0   ;I/O direction=output, 1-wire forced low
           movlw  167        ;167 counts (500  $\mu$ s)
           movwf  _cntr      ;into count register
RESET_P1  decfsz  _cntr      ;decrement count and skip next if=0
           goto   RESET_P1   ;else go back and decrement again
RESET_P2  bsf     TRISB,0   ;I/O direction=input, 1-wire pulled up
           bcf     STATUS,5  ;point back to Bank0
           movlw  20         ;20 count (60  $\mu$ s)
           movwf  _cntr      ;into count register
RESET_P3  decfsz  _cntr      ;decrement count and skip next if=0
           goto   RESET_P3   ;else go back and decrement again
           clrf   _temp      ;initialize samples register=0
RESET_P4  movlw  36         ;36 counts (180  $\mu$ s)
           movwf  _cntr      ;into count register
RESET_P5  btfss  PORTB,0   ;skip next if 1-wire is high
           incf   _temp      ;low so increment samples register
           decfsz _cntr      ;decrement count and skip next if=0
           goto   RESET_P5   ;else go back, sample, decrement again
           count 80 (240  $\mu$ s)
RESET_P6  movlw  80         ;into count register
RESET_P7  decfsz  _cntr      ;decrement count and skip next if=0
           goto   RESET_P7   ;else go back and decrement again
           return          ;done (temp returns sampled low count)
_W_B      rrf     _temp      ;rotate LSBit data into carry
           btfss  STATUS,0  ;skip next if carry (data)=1
           goto   W_B2      ;else jump to data=0 routine
W_B1      call   WIS        ;call the write-1 time slot
           goto   W_B3      ;go on
W_B2      call   WOS        ;call the write-0 time slot
W_B3      decfsz  _cntr      ;decrement count and skip next if=0
           goto   _W_B      ;else go back and do another bit
           return          ;done sending all bits
_R_B      call   R_S        ;call the read time slot
           movf   _rtemp     ;get the sampled low count
           bcf   STATUS,0    ;clear carry (read data=0 setup)
           btfsc STATUS,2   ;skip next if sampled count <>0
           bsf   STATUS,0    ;set carry (read data=1)
           rrf   _temp      ;rotate data into the MSBit
           decfsz _cntr      ;decrement count and skip next if=0
           goto   _R_B      ;else go back and do another bit
           return          ;done sending all bits
WIS       movlw  1          ;1 count (3  $\mu$ s)
           movwf  _ctemp     ;into count register

```

(continued)

Listing 1—continued

```

        bcf     PORTB,0    ;force 1-wire I/O bit low
        bsf     STATUS,5  ;point to Bank1 (direction control)
        bcf     TRISB,0   ;I/O direction=output, 1-wire forced low
W1S1   decfsz  _ctemp     ;decrement count and skip next if=0
        goto   W1S1      ;else go back and decrement again
        bsf     TRISB,0   ;I/O direction=input, 1-wire pulled up
        bcf     STATUS,5  ;point to Bank0
        movlw  33         ;33 count (99 µs)
        movwf  _ctemp     ;into count register
W1S2   decfsz  _ctemp     ;decrement count and skip next if=0
        goto   W1S2      ;else go back and decrement again
        return        ;done sending bit
WOS    movlw  30         ;30 count (90 µs)
        movwf  _ctemp     ;into count register
        bcf     PORTB,0   ;force 1-wire I/O bit low
        bsf     STATUS,5  ;point to Bank1 (direction control)
        bcf     TRISB,0   ;I/O direction=output, 1-wire forced low
WOS1   decfsz  _ctemp     ;decrement count and skip next if=0
        goto   WOS1      ;else go back and decrement again
        bsf     TRISB,0   ;I/O direction=input, 1-wire pulled up
        bcf     STATUS,5  ;point to Bank0
        movlw  04         ;4 count (12 µs)
        movwf  _ctemp     ;into count register
WOS2   decfsz  _ctemp     ;decrement count and skip next if=0
        goto   WOS2      ;else go back and decrement again
        return        ;done sending bit
R_S    movlw  1          ;1 count (3 µs)
        movwf  _ctemp     ;into count register
        bcf     PORTB,0   ;force 1-wire I/O bit low
        bsf     STATUS,5  ;point to Bank1 (direction control)
        bcf     TRISB,0   ;I/O direction=output, 1-wire forced low
R_S1   decfsz  _ctemp     ;decrement count and skip next if=0
        goto   R_S1      ;else go back and decrement again
        bsf     TRISB,0   ;I/O direction=input, 1-wire pulled up
        bcf     STATUS,5  ;point to Bank0
        movlw  3          ;3 count (15 µs)
        movwf  _ctemp     ;into count register
        cLrf  _rtemp     ;initialize sample count=0
R_S2   btfss  PORTB,0    ;skip next if 1-wire input=1
        incf  _rtemp     ;low so increment samples register
        decfsz _ctemp     ;decrement count and skip next if=0
        goto   R_S2      ;else go back and decrement again
R_S3   movlw  16         ;16 count (48 µs)
        movwf  _ctemp     ;into count register
R_S4   decfsz  _ctemp     ;decrement count and skip next if=0
        goto   R_S4      ;else go back and decrement again
        return        ;done receiving bit
ENDASM
END

```

Each device has a unique serial number associated with it, but when only one device is connected on the 1-wire bus, communication is simplified. The SKIP_ROM command (CCH) is the only command necessary to activate the device. SKIP_ROM commands the attached device to immediately wake up and take notice.

It's not necessary to know the ID number of the device. If you don't know the family of the device you're talking to (it may be any touch device), you can issue READ_ROM and the attached device will respond with its family code, ID number, and cyclic

redundancy check (CRC) byte. From the family code byte you can determine what kind of device is connected.

The 8-byte ROM code contains the family code byte, 6 bytes of ID code, and a CRC byte. The CRC byte is based

Figure 3—Here, I used a PicStic as a 1-wire interface controller. Calls to assembly-language routines from BASIC enable me to quickly write simple applications using 1-wire sensors. Output messages go to either a PC or a serial LCD.

on the previous 7 bytes and can be used to authenticate the validity of the 8-byte transmission. Although a wrong CRC byte might be because of a bad data transfer, it could also mean that multiple devices are talking at the same time.

This time around, I'm limiting the discussion to single devices tied to an I/O pin. Figure 2 shows how the CRC byte is calculated. Each bit shifted into the CRC byte is XOR'd with bits 4, 5, and 8 of the CRC before shifting.

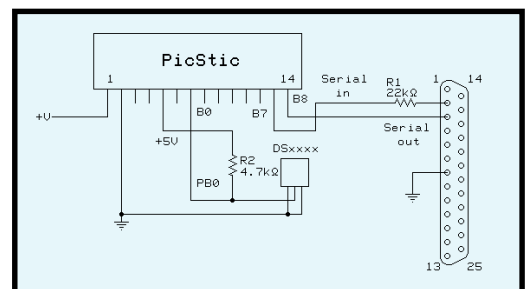
Although six assembly-language routines are shown in Listing 1—reset, write-1, write-0, read, write-byte, and read-byte—only the first four of these routines are necessary. With these four routines, you can make contact with almost any 1-wire device.

My first program was written as a generic program enabling you to experiment with various 1-wire devices. It was written for the PicStic using PicBasic but it can be used with almost any PIC microprocessor.

I like to use the PicStic because it contains all the essentials you need every time you build a PIC circuit—a 5-V regulator, crystal, and supporting capacitors. And because the PicStic is electrically reprogrammable (it uses a '16F84), my debugging time is reduced. PicBasic lets me embed assembly language where necessary and still use the higher level commands to quickly build the framework.

You'll also need to know the specifics for each 1-wire device (i.e., which commands to use for which devices and how many bytes a command will return, if any). Refer to the individual 1-wire datasheets for this information.

After wiring up the circuit in Figure 3, programming the PicStic, and applying power, you are presented with three choices—reset the 1-wire bus,



send a command, and request data.

Commands are broken down into function and memory commands. A reset is required before any function command, and a memory command may follow a function command. The function and memory commands are write only, but either command may offer a response.

Let's try using the program on a DS1820 connected to the 1-wire bus. Following these steps, we receive the responses displayed in Figure 4.

If the device's V_{CC} lead is connected to ground (true 1-wire connection) you must not use the 1-wire bus for 500 ms after sending a convert temp command (44h). Because the device uses parasitic power in this configuration, it must see a high for this duration.

After that, you can proceed with reading the conversion. However, if you connect its V_{CC} lead to V_{CC} , you can poll the device for end of conversion status with Response (bits-dec)? 8 <busy>.

The response to READ_SCRATCHPAD has the temperature in the first byte. If you know it's above 0°, you can stop reading after the first byte.

The conversation the master has with a 1-wire device can be terminated at any time. The temperature reply is in 0.5°C increments. To get a true temperature in Celsius degrees, divide this by 2 (ditch the 0.5 bit).

To convert this temperature to Fahrenheit, perform the conversion calculation:

$$F = \left(\frac{9C}{5}\right) + 32$$

or, in our case (because Celsius is in 0.5° increments):

$$F = \left(\frac{9C}{10}\right) + 32$$

This circuitry and software can easily be massaged into displaying the present temperature in both Celsius and Fahrenheit with a bit more software.

Using a serial LCD can eliminate the serial connection to your PC that would be used in the experimental and debugging process. Listing 2 shows you what my BASIC program looks like

```

RESET <presence>

Command (hex)? CC (SKIP_ROM)
Response (bits-dec)? 0

Command (hex)? BE (READ_SCRATCHPAD)
Response (bits-dec)? 72
<temp> <pos/neg> <user1> <user2> <res1> <res2> <cr> <cpc> <crc>
48 0 0 0 255 255 41 81 51

```

Figure 4—My first program was flexible enough to enable me to enter any of the 1-wire command bytes and receive responses from any of the devices connected. Here you can a SKIP_ROM command with no response and a READ_SCRATCHPAD command with a 9-byte (72-bit) response.

(minus the same assembly-language routines that are used in Listing 1).

Next month, I'm going to look at the 1-wire bus with multiple sensors. I'll show you how to get these devices to act civilly with one another.

You might want to check out Dallas's web site for documentation

on 1-wire devices. They have serial and parallel port dongles for reading external touch memory devices.

Software for reading these on the PC is also available, if you're into that. I prefer to provide my micros with the ability to use these devices directly. I hope that's your direction, too. 📧

Listing 2—This BASIC program (using assembly routines from Listing 1) communicates with a DS1820 1-wire digital thermometer and displays the temperature (in Celsius and Fahrenheit) on a PC or LCD.

```

'Written in PicBasic
'variable definitions
'b0 used in bit tests

symbol first=b5           ;Celsius value
symbol second=w3         ;b6 & b7 Fahrenheit value
symbol temp=b9           ;data being passed
symbol cntr=b10          ;bit counter
symbol ctemp=b11         ;temp counter
symbol rtemp=b12         ;temp zero counter

START: call RESET_P       ;1-wire reset (and presence)
if temp=0 then NO_GOT_P  ;check for no device
temp=$CC : cntr=8        ;SKIP_ROM command (it's 8 bits long)
call W_B                 ;1-wire send
temp=$44 : cntr=8        ;Convert Temperature command (it's 8 bits
                        ;long)
call W_B                 ;1-wire send
pause 500                ;wait for conversion
call RESET_P             ;1-wire reset (and presence)
if temp=0 then NO_GOT_P  ;check for no device
temp=$CC : cntr=8        ;SKIP_ROM command (it's 8 bits long)
call W_B                 ;1-wire send
temp=$BE : cntr=8        ;READ_SCRATCHPAD (it's 8 bits long)
call W_B                 ;1-wire send
cntr=8                   ;8 bits
call R_B                 ;1-wire read
first=temp/2             ;first=value read (divide by 2 for nearest
                        ;whole degree)
second=temp*9            ;here's where the conversion
second=second/10        ;to degrees F
second=second+32        ;is done
serout 7, n2400,(12)    ;form feed
serout 7, n2400,("The Temperature is...",13,10)
serout 7, n2400,(13,10,#first," Celsius",13,10,#second," Fahrenheit")
goto START              ;do it all again...

NO_GOT_P:
serout 7, n2400,("No device",13,10)
goto START

```

Jeff Bachiochi (pronounced "BAH-key-AH-key") is an electrical engineer on Circuit Cellar INK's engineering staff. His background includes product design and manufacturing. He may be reached at jeff.bachiochi@circuitcellar.com.

SOFTWARE

Source code for this article, as well as a Dallas 1-wire parts list, may be found via the Circuit Cellar web site.

SOURCES

1-wire devices

Dallas Semiconductor
(800) 336-6933
(972) 371-4000
Fax: (972) 371-3715
www.dalsemi.com

PicStic

Micromint, Inc.
(800) 635-3355
(860) 871-6170
Fax: (860) 872-2204
www.micromint.com

PicBasic

microEngineering Labs
(719) 520-5323
Fax: (719) 520-1867
www.melabs.com