



Column #82 February 2002 by Jon Williams:

## Weather On the Wire

*I've always had an interest in the weather. As a boy I would read books on basic meteorology and simple weather forecasting. There was a moment, albeit brief, when I thought about being a TV weather man. After high school and a couple of years in the military, I went to work in the irrigation industry and spent most of that time in the golf course business – a business tied very directly to the weather.*

Even though I've been out of that business for a couple of years, my morning routine is the same: I wake, I stretch, I go out onto the front porch to see what the weather is like and what I can expect for the day. Today (early January as I write) was really nice. It's clear in Dallas and the temperature on my front porch was 65 degrees and the air is clear. You can't beat that.

A couple of months ago I was contacted by a Stamp programmer named Tim. He lives in Tucson and has a big interest in the weather – very big. He told me about a low-cost weather station that used Dallas Semiconductor 1-Wire technology. I was under the impression that these stations were no longer available, but I was wrong. A company called AAG in Mexico has them and you can order them via the Internet using PayPal. A 1-Wire weather station seems like a neat add-on for a BS2p, so I bought a few.

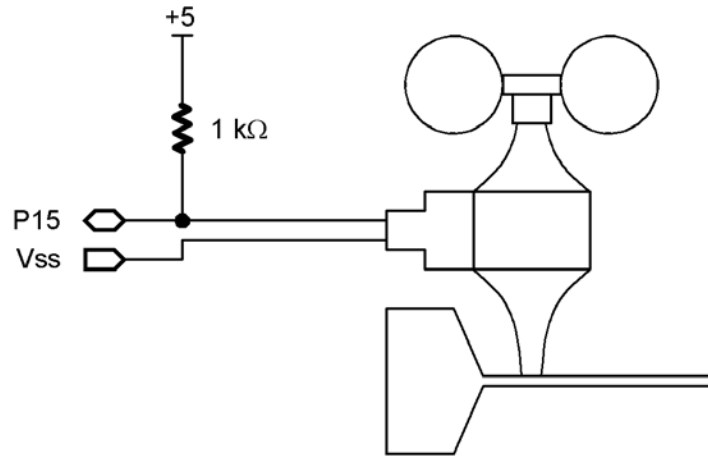
**Figure 82.1: Dallas Semiconductor 1-Wire Weather Station**



### **1-Wire Weather Station**

Since we don't have to build the station, I'm not going to include the schematic here. It is available on the AAG web site if you'd like to review it as we go through the code. The key thing to understand is that there are three devices connected to the 1-Wire bus: a DS1820 thermometer, a DS2423 counter for the anemometer (wind speed) and a DS2450 quad analog-to-digital converter that is connected to the wind vane.

Figure 82.2: 1-Wire Weather Station BS2p Schematic



The connection to the weather station is an RJ-11 (telephone) connector. If you have a BS2p Demo Board this project is easy since there is an RJ-11 on the board. Even if you're making your own connections, please note that the 1-Wire pull-up is very stiff, 1 kOhm. This is necessary to overcome the impedance of the cable connecting the Stamp to the weather station. If you are using the Demo Board, remove jumper B2 and connect a 1 kOhm pull-up to the 1-Wire bus.

### Weather Code

With nothing to build, let's jump right into the code. Now ... before we even start we need to know what the 1-Wire serial numbers are. I've worked with two of these stations and neither came with the serial numbers printed on the side of the box (nor was I expecting it). This is not a problem. What we need to do is run the 1-Wire SearchRom code we developed last April. If you don't have it, you can download the code from the Nuts & Volts web site.

When you run SEARCHDEMO.BSP you'll see that it doesn't know what a DS2423 is and lists it as an "Unknown Device." Don't worry about this. Just copy the serial numbers so that they can be plugged into the weather station code.

Figure 82.3: BS2p Debug Screen with 1-Wire Code Running



Okay, this listing is pretty long so we better jump right into it. The Initialization section opens the **DEBUG** window, displays a header for the screen the gets down to work. The first piece of real work is to make sure that the 1-Wire bus is not shorted. This is done by making the 1-Wire comm pin an input then reading it to make sure we see it pulled up to +5 (will read 1). If zero is returned, the program is aborted to an error handler.

The next section of code is the reason we just did that check. There's a small power supply in the weather station. Okay, okay, it's just a diode and a capacitor. The cap provides Vdd to the 1-Wire devices and to pull-ups connected to the A2D converter. Since the supply "borrows" power from the 1-Wire bus, we start the program by charging it. This is done by making the 1-Wire bus pin

high and waiting for ten seconds. In our program a message is displayed showing the charging time left.

Once the station is charged, labels for communication status and each of the sensors is printed. The program then enters the main loop. In the main loop, the program checks to see that the station is connected the branches to the current sensor. So long as the station stays connected, the program will run the main loop. If the station gets disconnected, the initialization sequence will be re-run until contact with the station is reestablished.

We can check to see if the station is connected by issuing the SearchROM command and reading back two bits. If nothing is connected all we'll see is the pulled-up bus and a value of %11. If there are devices connected we'll get another pattern. The reason for checking is the same as looking for a short: there's probably a lot of wire between your desk and the station and accidents can happen. We'll assume that the world is a perfect place for wires and that there are not shorts and the station is connected. Now we can read and display the sensors.

### Temperature

The first sensor is the DS1820 temperature sensor. We've used this before so we'll brief in the coverage. The routine starts by loading the device serial number into an array called *romData*. This process is facilitated by a small sub-routine called **Load\_SN**. What we'll do is load the **DATA** address of the device into *eeData*, then call this routine. The serial number is read out of EEPROM and place into the array.

With the serial number loaded, we'll issue a MatchROM command with the serial number, then the temperature conversion command. All of this happens in a single **OWOUT** statement. Then the 1-Wire bus pin is taken high and we wait for 750 milliseconds. This is the worst-case conversion time for the DS1820 and making the bus high provides power during the conversion process.

The 9-bit Celsius temperature is retrieved with **OWIN** and converted to Fahrenheit for display. Note that the temperature is a signed value. This time of year the temperature in many areas of the country fall below zero Celsius (32 Fahrenheit).

The temperature is displayed using a subroutine, **RJ\_Print**, developed by Stamp guru, Stamp list contributor and all-around-nice-guy, Dr. Tracy Allen. To use this routine, we need to pass the width out of field in the variable *width* and our value to print in *rjNum*. Tracy's routine makes clever use of **LOOKDOWN** to determine how many spaces to print (pad) before the actual number. If the number is negative a minus sign will be printed in front of the number.

## Wind Speed

The 1-Wire weather station measures wind by counting the number of pulses during a known time window. Mechanically, there is an arm connected to the anemometer cups that carries a couple of magnets to trip a reed switch connected to the DS2423 counter. Since the counter cannot be reset, what we'll have to do is get the current count, delay for some period, read the count again and then subtract the first reading from the second to obtain difference.

What Tim noticed and I confirmed is that the counter only seems to work properly when used in overdrive mode. Neither one of us can explain it, but I wanted to make you aware in case you decide to make modifications to the program.

To put the DS2423 in overdrive mode, we send the ODMatchROM value by itself. After that, we send additional data using the high speed setting of the **OWOUT** command. Since the DS2423 is actually a memory plus counter, we're forced to read the final byte of the page that precedes the counter value. It's a quirk of the DS2423 that you cannot reset the counter (it is reset on power-up) nor read it directly. There are actually two independent counters in the DS2423. The weather station ties the inputs together so reading either will give us the same value.

Our **OWIN** ends up reading the lower 16 bits of a 32-bit counter. Then the program pauses for 2500 milliseconds and reads the counter again. The difference is computed and the wind speed is calculated.

The general formula for wind speed is given as:

$$\text{Speed (mph)} = \text{Revolutions\_Per\_Second} * 2.453$$

There are two magnets on the t-bar that rotates above the wind speed counter switch, so each revolution actually gives us two counts. If we paused for one second, our wind speed resolution would be 1.23 mph (2.453 mph / 2). Since I wanted a little better resolution than that, I used a **PAUSE** value of 2500. This brings the resolution down to 0.49 mph – much better. What this means, though, is we have to wait for the wind speed reading. This is the reason the sensor indication was developed.

In order to maintain our resolution within the Stamp's integer math system, what we're actually going to use is 24.53 to get tenths. Since our **PAUSE** value is 2500 – equivalent to waiting five seconds if one revolution equaled one count -- we divide that by five and get 4.906. For use with the Stamp we multiply 4.906 by 256 and use the \*/ (star-slash) operator. The result is multiplication by a fractional value.

The variable *speed* now holds the wind speed in tenths of a mile per hour. For display, we extract the whole part by dividing the wind speed value by ten, then sending it off to **RJ\_Print**. Then we print a decimal point and the tenths digit by using the **DEC1** modifier.

### Wind Direction

While many weather stations use a free-rotating potentiometer to determine wind direction, the 1-Wire station takes a different approach. It uses eight reed switches connected to a resistor array that is read by an DS2450 quad A2D converter. The way the resistors are connected means that a given A2D channel will either see full scale, half-scale or ground. Our trick will be to convert these four analog readings into a 1-of-16 direction value.

The DS2450 is configured to read all four channels, using 8-bit values with a 5.12 volt reference. Since the actual voltage at the station is going to be a bit less than five volts, our highest reading will be slightly less than 255 (full scale at 5.12 volts). I wrote a quick test program to view and decode the values for each of the 16 compass directions. High values tended to be around 235 to 240, middle values around 120 and low values between 0 and 3.

To get the A2D data, we issue a ReadMem command starting at address 0. We need retrieve eight bytes of data since the DS2450 can be configured for 16-bit values. Since we don't need the *romData* array anymore, the A2D data is read into it. Next we'll grab every other byte to create a single word value that can be used in a table to determine direction.

As each A2D channel is retrieved from *romData* it is divided by 90. With integer math, this division will always return a value of two (high readings), one (middle readings) or zero. The result of each division is placed into a separate Nib in the variable *direction*. At this point *direction* holds a value that tells us the direction from which the wind is blowing. **LOOKDOWN** is used to convert this value into a zero to 15 value that can be used to point to a string. You can see how the large lookup table was split in two pieces and that the variable *dFlag* is set to an out-of-range value before **LOOKDOWN** is used.

If the value returned in *direction* is invalid (it happens – one of my stations has a bad part) the value of *dFlag* will not be changed by **LOOKDOWN**. This can be used to flag an error as we've done with this program. If value was good, the **LOOKDOWN** tables will convert it to a value between zero and 15. We use this value to calculate the address of the string that tells us the direction and print it with the subroutine called **Print\_String**.

**That's A Wrap**

Well, that about wraps it up for this month. For those of you interested in the weather this should get you started. Be sure to take a look at Tim's site and the Dallas Semiconductor site for more sensors.

Next month ... GPS. Yes, I said I wouldn't write about it but I've been convinced that I should. We'll be using the BS2p again and taking advantage of the SPSTR modifier of **SERIN**.

Until then, Happy Stamping.

**(Additional) Resources**

Tim Bitson's Weather Page  
[www.timbitson.com/weather/index.html](http://www.timbitson.com/weather/index.html)

AAG  
[www.aag.com.mx](http://www.aag.com.mx)

```

' -----[ Title ]-----
' Program Listing 82.1
' File..... OW WEATHER.BSP
' Purpose... 1-Wire Weather Station Interface
' Author.... Jon Williams / Parallax (portions by Tim Bitson)
' E-mail.... jonwms@aol.com
' Started... 13 DEC 2001
' Updated... 06 JAN 2002

' { $STAMP BS2p }

' -----[ Program Description ]-----
'
' This program reads and displays the sensors from a Dallas 1-Wire Weather
' Station.
'
' NOTE: The 1-Wire bus requires a very stiff pull-up -- 1 kOhm or less.

' -----[ Revision History ]-----
'
' 15 DEC 2001 - Version 1 working
' 06 JAN 2002 - Updated error handling for bad wind vane

' -----[ I/O Definitions ]-----
'
OWpin          CON      15          ' 1-Wire bus

' -----[ Constants ]-----
'
' 1-Wire Support
'
OW_NoRst       CON      %0000      ' no Reset
OW_FERst       CON      %0001      ' Front-End Reset
OW_BERst       CON      %0010      ' Back-End Reset
OW_BitMode     CON      %0100
OW_HighSpd     CON      %1000

ReadROM        CON      $33         ' read ID, serial num, CRC
MatchROM       CON      $55         ' look for specific device
ODMatchROM     CON      $69         ' overdrive match rom
SkipROM        CON      $CC         ' skip rom (one device)
SearchROM      CON      $F0         ' search

' DS1820 control
'
CnvrtTemp      CON      $44         ' do temperature conversion

```

**Column #82: Weather On the Wire**

```

RdScratch      CON      $BE          ' read scratchpad
' DS2423 control
'
ReadMemCntr    CON      $A5          ' read memory + counter
' DS2450 control
'
ReadMem        CON      $AA          ' read memory
WriteMem       CON      $55          ' write memory
Convert        CON      $3C          ' do conversion

NoDevice       CON      %11          ' no device present
CommOkay      CON      1
NoComm         CON      0

Celsius        CON      0
Fahrenheit     CON      1
TMode          CON      Fahrenheit  ' temperature display mode
DegSym         CON      176

MoveTo         CON      2            ' DEBUG cursor positioning
LF             CON      10           ' linefeed

' -----[ Variables ]-----
'
devCheck       VAR      Nib          ' device check return ocode
commStat       VAR      Bit          ' comm status
sensor         VAR      Nib          ' current sensor to display
eeAddr         VAR      Word         ' string pointer in EE
char           VAR      Byte         ' character to print
idx            VAR      Nib          ' loop counter
romData        VAR      Byte(8)      ' ROM data to devices

tempIn         VAR      Word          ' raw temperature
sign           VAR      tempIn.Bit11 ' 1 = negative temperature
tLo           VAR      tempIn.LowByte
tHi           VAR      tempIn.HighByte
tSign         VAR      Bit
tempC         VAR      Word          ' Celsius
tempF         VAR      Word          ' Fahrenheit

cntrA          VAR      Word          ' wind count start
cntrB          VAR      tempIn        ' wind count end
rps            VAR      tempC         ' revs per second
speed          VAR      tempF         ' wind speed

crc16          VAR      tempIn
verify         VAR      char
direction      VAR      tempC

```

```

dFlag          VAR      tempF.LowByte

rjNum          VAR      Word           ' number to right justify
width         VAR      Nib           ' width of field
rjSign        VAR      Bit
digits        VAR      Nib

' -----[ EEPROM Data ]-----
,
DS1820        DATA    $10,$15,$32,$09,$00,$08,$00,$A9      ' temp sensor
DS2423        DATA    $1D,$82,$43,$01,$00,$00,$00,$6F      ' wind speed
DS2450        DATA    $20,$AF,$FF,$00,$00,$00,$00,$5E      ' wind direction

CommUp        DATA    "... Up", 0
CommDn        DATA    "... Down", 0

WindDir       DATA    " N", 0, " NNE", 0, " NE", 0, " ENE", 0
               DATA    " E", 0, " ESE", 0, " SE", 0, " SSE", 0
               DATA    " S", 0, " SSW", 0, " SW", 0, " WSW", 0
               DATA    " W", 0, " WNW", 0, " NW", 0, " NNW", 0

BadDir        DATA    "Error", 0

' -----[ Initialization ]-----
,
Initialize:
  DEBUG CLS                               ' open DEBUG window
  PAUSE 250

Splash Screen:
  DEBUG Home
  DEBUG "=====", CR
  DEBUG " BS2p <--> 1-Wire Weather Station ", CR
  DEBUG "=====", CR

Charge Station:
  INPUT OWpin
  IF (Ins.LowBit(OWpin) = 0) THEN Bus_Shorted ' possible wiring problem

  HIGH OWpin
  DEBUG MoveTo, 0, 4, "Charging station..."
  FOR idx = 10 TO 1
    DEBUG MoveTo, 20, 4, DEC idx, " "
    PAUSE 1000
  NEXT
  INPUT OWpin
  DEBUG MoveTo, 0, 4, REP " "\25

```

## Column #82: Weather On the Wire

```
Weather Labels:
  DEBUG MoveTo, 0, 4, "Comm..... "
  DEBUG MoveTo, 0, 6, "( ) Temperature... ", CR
  DEBUG MoveTo, 0, 7, "( ) Wind Speed.... ", CR
  DEBUG MoveTo, 0, 8, "( ) Direction..... ", CR

' -----[ Main Code ]-----
,
Main:
  GOSUB Comm_Status                ' check 1-Wire comm
  IF (commStat = CommOkay) THEN Show_Sensors
  PAUSE 2000
  GOTO Initialize                  ' "reboot" if comm went down

Show Sensors:
  GOSUB Flag_Sensor                ' show sensor in use
  BRANCH sensor, [Show_Temp, Show_Wind_Speed, Show_Wind_Dir]

Main2:
  sensor = sensor + 1 // 3         ' select next sensor
  GOTO Main
  END

' -----[ Subroutines ]-----
,

' *****
' A shorted condition on the 1-Wire bus has been detected
' -- we can't charge the station cap in this condition
' *****
,

Bus Shorted:
  DEBUG CLS, "The 1-Wire bus is shorted.", CR
  DEBUG "Please repair before continuing.", CR
  END

' *****
' Print status of 1-Wire comms
' *****
,

Comm Status:
  eeAddr = CommUp                  ' assume bus is up
  commStat = CommOkay
  GOSUB Device_Check              ' check bus for devices
  IF (devCheck <> NoDevice) THEN Print_Comm
  eeAddr = CommDn
  commStat = NoComm
```

```

DEBUG MoveTo, 19, 6, REP " "\15          ' clear station data if down
DEBUG MoveTo, 19, 7, REP " "\15
DEBUG MoveTo, 19, 8, REP " "\15

Print Comm:
  DEBUG MoveTo, 12, 4                    ' show comm status
  GOSUB Print_String
  RETURN

' *****
' Check for presence of 1-Wire devices
' -- does not search for ROM codes
' *****
'
Device Check:
  devCheck = 0
  OWOUT OWpin, OW_FERst, [SearchROM]
  OWIN OWpin, OW_BitMode, [devCheck.Bit1,devCheck.Bit0]
  RETURN

' *****
' Indicate sensor in use
' *****
'
Flag Sensor:
  DEBUG MoveTo, 1, 6, " "                ' clear previous mark
  DEBUG MoveTo, 1, 7, " "
  DEBUG MoveTo, 1, 8, " "
  DEBUG MoveTo, 1, (6 + sensor), "*"    ' mark current sensor
  RETURN

' *****
' Transfer OW serial number to RAM
' -- point to SN with eeAddr
' *****
'
Load SN:
  FOR idx = 0 TO 7                      ' load ROM pattern
    READ (eeAddr + idx), romData(idx)
  NEXT
  RETURN

' *****
' Retrieve and display temperature
' *****
'
Show_Temp:

```

## Column #82: Weather On the Wire

```
eeAddr = DS1820                ' load device serial number
GOSUB Load_SN

OWOUT OWpin, OW FERst, [MatchROM, STR romData\8, CnvrtTemp]

HIGH OWpin                      ' extra juice during conversion
PAUSE 750
INPUT OWpin

OWOUT OWpin, OW FERst, [MatchROM, STR romData\8, RdScratch]
OWIN OWpin, OW BERst, [tLo, tHi]

tSign = sign                    ' save sign bit
tempIn = tempIn >> 1          ' round to whole degrees
IF (tSign = 0) THEN NoNeg1
tempIn = tempIn | $FF00        ' extend sign bits for negs

NoNeg1:
tempC = tempIn                ' save Celsius value
tempIn = tempIn * / $01CC     ' multiply by 1.8
IF (tSign = 0) THEN NoNeg2    ' if neg, extend sign bits
tempIn = tempIn | $FF00

NoNeg2:
tempF = tempIn + 32           ' finish C -> F conversion

DEBUG MoveTo, 19, 6           ' move cursor to temp display
rjNum = tempF                 ' prep for right justified print
width = 5                     ' five digits wide
GOSUB RJ_Print
DEBUG " ", DegSym, "F"
GOTO Main2

' *****
' Retrieve and display wind speed
' -- mph = rps * 2.453
' *****
'

Show Wind Speed:
eeAddr = DS2423
GOSUB Load_SN

' get starting count
'
OWOUT OWpin, OW FERst, [ODMatchROM]
OWOUT OWpin, OW HighSpd, [STR romData\8, ReadMemCntr, $DF, $01]
OWIN OWpin, (OW_HighSpd | OW_BERst), [cntrA.LowByte, cntrA.LowByte,
cntrA.HighByte]

PAUSE 2500                    ' = revs for 5 seconds
```

```

GOSUB Comm_Status
IF (commStat = CommOkay) THEN Get_End_Count
GOTO Main2                                ' abort if comm down

' get ending count
,
Get_End_Count:
OWOUT OWpin, OW FERst, [ODMatchROM]
OWOUT OWpin, OW HighSpd, [STR romData\8, ReadMemCntr, $DF, $01]
OWIN OWpin, (OW HighSpd | OW BERst), [cntrB.LowByte, cntrB.LowByte,
cntrB.HighByte]

Calc_CPS:
rps = cntrB - cntrA                        ' get differential count
IF (cntrB >= cntrA) THEN Calc_Speed
rps = rps + $FFFF                          ' correct for rollover

Calc_Speed:
speed = rps * / $04E7                      ' rps * 4.906 (tenths @ 5 seconds)
DEBUG MoveTo, 19, 7
width = 3                                  ' width of whole portion
rjNum = speed / 10                         ' get whole portion
GOSUB RJ_Print                             ' print it
DEBUG ".", DEC1 speed, " mph "            ' then print tenths
GOTO Main2

' *****
' Retrieve and display Wind direction
' *****
,
Show Wind Dir:
eeAddr = DS2450
GOSUB Load_SN
OWOUT OWpin, OW FERst, [MatchROM, STR romData\8, WriteMem, $08, $00]

FOR idx = 1 TO 4                            ' setup four channels
OWOUT OWpin, OW NoRst, [$08]                ' 8-bit values
OWIN OWpin, OW NoRst, [crc16.LowByte, crc16.HighByte, verify]
OWOUT OWpin, OW NoRst, [$01]              ' 5.12 volt scale
OWIN OWpin, OW NoRst, [crc16.LowByte, crc16.HighByte, verify]
NEXT

Start Conversion:
OWOUT OWpin, OW FERst, [MatchROM, STR romData\8, Convert, $0F, $55]
OWIN OWpin, OW NoRst, [crc16.LowByte, crc16.HighByte]

HIGH OWpin                                  ' juice during conversion
PAUSE 10
INPUT OWpin

```

## Column #82: Weather On the Wire

```
Get_A2D:
  OWOUT OWpin, OW FERst, [MatchROM, STR romData\8, ReadMem, $00, $00]
  OWIN  OWpin, OW BERst, [STR romData\8]

Calc_Direction:
  FOR idx = 0 TO 3
    direction.LowNib(idx) = romData(idx * 2 + 1) / 90
  NEXT

  ' convert word to 1 of 16

  dFlag = 99
  LOOKDOWN direction, [$2220,$1220,$1222,$1122,$2122,$2112,$2212,$2211], dFlag
  IF (dFlag < 8) THEN Show_Direction
  ' found in first table

  LOOKDOWN direction, [$2221,$0221,$0222,$0022,$2022,$2002,$2202,$2200], dFlag
  dFlag = dFlag + 8
  ' adjust for second table

Show_Direction:
  DEBUG MoveTo, 19, 8
  eeAddr = BadDir
  IF (dFlag < 16) THEN Good_Dir
  GOTO Dir_To_Screen
  ' found in either table?

Good_Dir:
  eeAddr = WindDir + (6 * dFlag)
  ' address of direction string

Dir_To_Screen:
  GOSUB Print_String
  GOTO Main2

' *****
' Print a right-justified number
' *****
'

RJ_Print:
  rjSign = rjNum.Bit15
  rjNum = ABS(rjNum)
  digits = width
  LOOKDOWN rjNum, <[0,10,100,1000,65535], digits
  DEBUG REP " "\ (width-digits-1), 13 * rjSign + " ", DEC rjNum
  RETURN

' *****
' Print string at current cursor position
' -- point to string with eeAddr
' -- string is zero-terminated
' *****
'
```

```
Print_String:
  READ eeAddr, char
  IF (char = 0) THEN Print String Done
  DEBUG char
  eeAddr = eeAddr + 1
  GOTO Print_String

Print String Done:
  RETURN
```