OWL2pe and BASIC Stamp to anemometer

<stamp index> <home>

Contents (updated 31/2007)

Anemometer (rotating cups with reed relay signal)

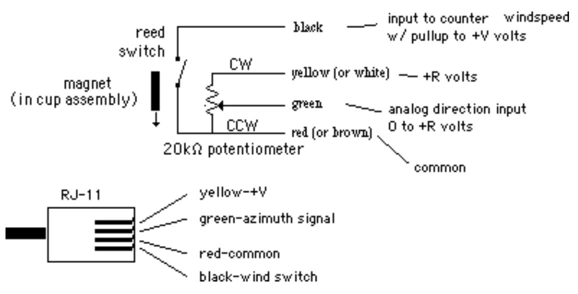
top

Here is calibration data from a couple of popular anemometers:

Davis anemometer, models 7911, 7913, 7914, and 6410 (vantage pro):

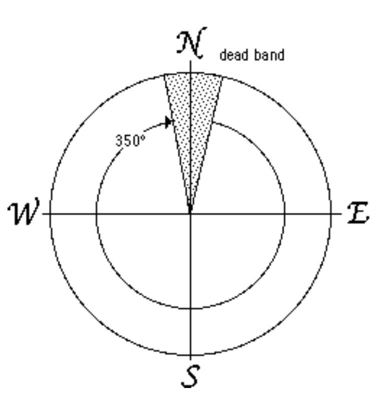
- one pulse per revolution of the cups, a magnetic reed switch
- 2.25 mile-per-hour per hertz
- 0.44444 hertz (pulses per second) per mph.
- sixty hertz at 135 mph windspeed.
- Starting threshold is 2-4 mph

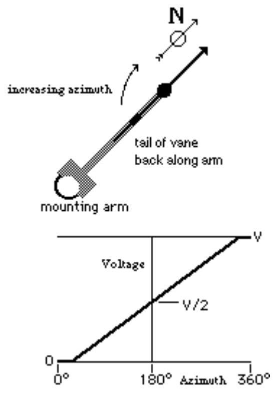
mph = 2.25 * Hertz
mps = 1.006 * Hertz



Note, the above data comes from recent conversations with Davis. The display console from Davis Instruments update the wind display once every 2.25 seconds. In recent versions, there is a 100 ohm resistor in series with the reed switch (which protects it from large currents if someone accidentally attaches the switch directly to a power supply)

The wind vane is a potentiometer of nominal value 20 kohms, +/- 1 kohm. The position of the wiper, or the resistance from wiper to end terminal is linear with azimuth. However, there is a dead band of around 10 degrees. As shipped from the factory, the crossover point occurs when the tail of the vane is back along the arm and the tip of the vane points out from the arm. That direction is nominally North.





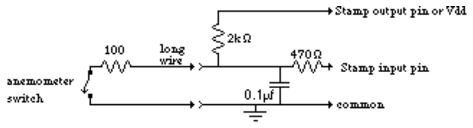
Met One 034B calibration factors:

- 0.8 meters-per-second per hertz (= 1.789 miles-per-hour per hertz)
- 60 hertz at 48 meters-per-second (= 107.4 mph)

mph = 1.789 * Hertz
mps = 0.8 * Hertz

Interface using the COUNT command

The anemometer is connected to an i/o pin on the OWL2 or BASIC Stamp with a pullup resistor (~2k to 20k).



Use the higher resistance value when the wiring to the anemometer is short and the lower value when the wiring is long or subject to leakage or noise. The pullup resistor can go either to a power supply or to to another Stamp i/o pin. The advantage of using another Stamp pin is that it can be made high only when it is time to count pulses, to save power if need be. Note that the anemometer switch can stop anywhere in its travel, with the switch either open or closed. The 100 ohm resistor is built into the DV6410 anemometer, and it protects the reed switch from overcurrent that would cause sparking and shorten its life. The 470 ohm resistor is there to protect the Stamp pin from ESD.

We'd like to count the number of pulses in a time interval, which is directly proportional to windspeed. Counting the signal from the Davis 6410 for 2.25 seconds will give a result directly in units of mph,

```
     mph = 2.25 * counts / seconds     ' (so counts=mph when seconds=2.25)
```

Here is how to implement this on the **OWL2pe** (Stamp BS2pe) data logger:

```
  wind var byte
  COUNT wpin,3125,wind     ' the BS2pe marks time in units of 0.72 milliseconds, so 2250 mS / 0.72 mS per unit = 3125 units
```

The syntax of the BASIC Stamp COUNT command includes the duration parameter, which on the original BS2 and on the BS2e is one millisecond units. But it is different on the other Stamps, 0.72 mS on the BS2pe, 0.4mS on the BS2sx and 0.287 mS on the BS2pe. So the parameter that goes in the units slot for those stamps is different, always to get the 2250 mS total count time.

Sometimes windspeed must be returned directly in other units. Meters per second and miles per hour are the most common units.

```
   mps = mph * 0.447
   mph = mps * 2.237
```

Here are other COUNT commands useful with the Davis anemometers:

```
  wind var byte
  COUNT wpin,3125,wind     ' BS2pe, 1 mph resolution with Davis, count for 2.25 seconds.
  DEBUG "mph=",DEC wind,CR

  COUNT wpin,6250,wind     ' BS2pe, 0.5mph resolution with Davis, count for 4.5 seconds.
  DEBUG "mph=",DEC wind*5/10,".",DEC1 wind*5,CR

  COUNT wpin,1397,wind     ' BS2pe, 1 mps resolution with Davis, count for 1.006 seconds
  DEBUG "mps=",DEC wind,CR

  COUNT wpin,2794,wind     ' BS2pe, 0.5 mps resolution with Davis, count for 2.012 seconds
  DEBUG "mps=",DEC wind*5/10,".",DEC1 wind*5,CR
```

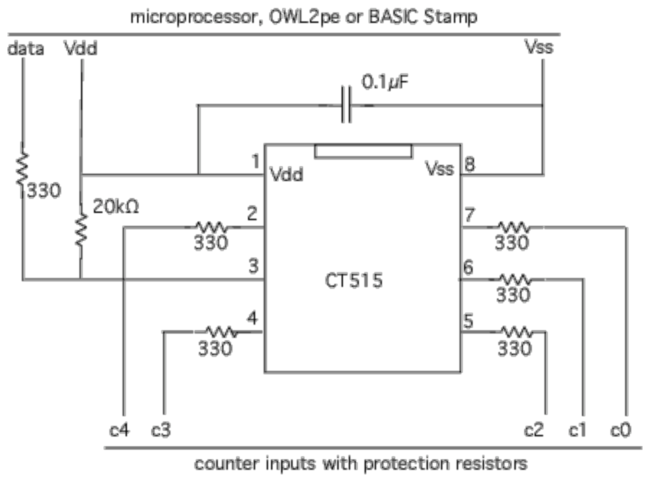
It may be convenient to count for some constant length of time and then convert mathematically to units of windspeed.   For example, counting for 5 seconds:

```
  wind var byte
  COUNT wpin, 1389*5, wind   ' BS2pe, count for 5.0 seconds, 1389 is 1 second on the 'pe
  wind = wind */ 5760 / 5     ' */576 is stampese for *2.25
  DEBUG "mph=",DEC wind/10,".",DEC1 wind   ' display as xx.x
```

Interface using an external CT515 counter chip:

One disadvantage of the above approach is that the OWL2pe or BASIC Stamp has to stay awake while counting for a period of several seconds. That wastes power in a battery or solar powered system and it also occupies the processor so that it can't be doing other things at the same time.   This is especially a problem when several anemometers or other pulse frequency type devices need to be tracked.

The CT515 (CT515.PDF) is a support chip for the OWL2pe or BASIC stamp designed at EME Systems that takes over the task of counting events, such as the pulses from an anemometer, rain gage, flow meter, or traffic counter. The CT515 monitors 5 event channels in parallel while consuming very little power supply current.   It includes contact debouncing.


microprocessor, OWL2pe or BASIC Stamp
data
Vdd
Vss
0.1µF
330
20kΩ
1
Vdd
Vss
8
2
330
7
330
3
CT515
6
330
4
330
5
330
c4
c3
c2
c1
c0
counter inputs with protection resistors

The anemometer is attached to one of the input channels of the CT515. Other pulse type devices, such as a rain gage or flow meter or geiger counter, or another anemometer can be connected to the other 4 CT515 channels. Periodically, perhaps at an interval of 10 seconds, the OWL2pe reads out the accumulations present on all 5 channels.

```
     LOW ct515data
     PAUSE 10
     INPUT ct515data
     SERIN ct515data, $54, [SPSTR 12] ' 1 status, and 5 data words,
```

Supposing the anemometer is attached to counter input c0, the wind count for 10 seconds is now available in the scratchpad locations 2 and 3.

```
     x VAR Byte
```

```
GET 2, word x
x = x */ 5760 / interval    ' */ 576 is stampese for 2.25 mph per hertz (576/256=2.25)
DEBUG "mph=",DEC wind/10,".",DEC1 wind   ' display as xx.x
```

Please refer to the CT515 document for advice on how to set up and read out the CT515 data.

Further computations:

Often the wind speed data needed will be the average for some period of time, and the maximum over that same time, or other statistics. The following routine has one part that executes frequently, and another part that executes only when data is logged into the file:

```
 wind     VAR   Byte
 windacc  VAR   Word    ' accumulation for average
 windmax  VAR   Byte    ' maximum
 windmin  VAR   Byte    ' minimum
 Nsmpl    VAR   Word    ' number of samples
 windmin = 255

' frequently, e.g. once per 10 seconds
  COUNT wpin,2250,wind    ' windspeed to one mile per hour resolution
  windmax = wind MIN windmax ' maintains maximum windspeed
  windmin = wind MAX windmin ' maintains minimum windspeed (there is not a typo, MAX does for minimum)
  windacc=windacc + wind     '
  Nsmpl=Nsmpl+1              ' one more sample

' infrequently, only when it is time to log data
  result=windacc/Nsmpl       ' compute average
  GOUSB logdata              ' log it
  result=windmax             ' fetch maximum
  GOSUB logdata              ' log it
  windavg=0                  ' reinitialize
  Nsmpl=0
  windmax=0
  windmin=255
```

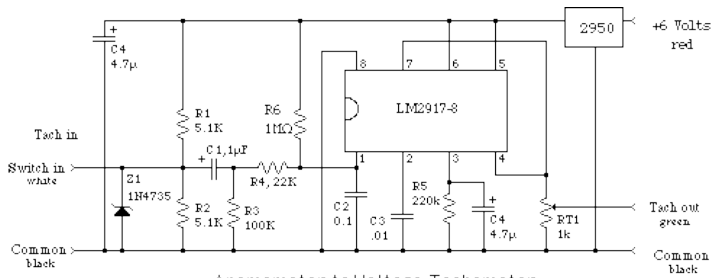
Windspeed tends to be sporadic, and the distribution highly skewed. Infrequent samples (e.g. hourly) of windspeed tend to be quite low, compared to the average or maximum. It is best to sample the windspeed frequently. The sum may need to use double precision math (32 bits) if lots of samples are taken, enough to overflow the accumulator.

Here is our double precision averaging routine for t he OWL2pe. This routine expects the new raw value in a word variable wx, and the variable wpe1 is a pointer to a 4 byte buffer in the scratchpad that holds the current accumulation, and the variable wj holds the current number of samples. At exit, the current average value (accumulation over number of samples) is returned in the variable wx, the scratchpad buffer contains the new accumulation, and wj is unchanged. The routine uses a couple of other variables (wy and wz) for intermediate computations.

```
Average4:
 ' Double Precision averaging routine
 ' enter with
 ' wpe1 = address of 2 words (4 bytes) allocated in SPram
 ' wj = number of samples
 ' wx = value to accumulate
 ' exit with
 ' wx = current average value result
 ' wj and wpe unchanged
 ' uses wy and wz for computation
   GET wpe1,Word wy      ' get low word of accumulation
   wy=wy+wx              ' accumulate
   PUT wpe1,Word wy      ' save
   GET wpe1+2, Word wz   ' high word of accumulation
   IF wy<wx THEN         ' accumulate iff carry from low word
     wz=wz+1
     PUT wpe1+2,Word wz
   ENDIF
 ' now double precision divide by # of samples
   wx=65535//wj*wz//wj+wz+(wy//wj)
   wx=65535/wj*wz+(65535//wj*wz/wj)+(wy/wj)+(wx/wj)
   RETURN
```

Wind interface using an analog Tachometer

Here is a circuit that can convert the output of the Davis anemometer (Or any anemometer that uses a switch-type output), into a voltage for input to an analog to digital converter. The output of this circuit can be calibrated to give 1 volt output at 100 miles per hour by adjusting RT1. The network of resistors and capacitors at the input is there to turn the on-off switch action into a series of bipolar pulses for input into the LM2917 tachometer chip (National Semi). Also, the circuit at the input protects the circuit from lightning induced surges. Anemometers are available that have a AC generator, and these produce an AC sine wave signal. Those anemometers can also use this circuit, but remove R1, R3 and R6 and replace C1 with a jumper.



Anemometer to Voltage Tachometer.
Tracy Allen, EME Systems, (c) 1992

Interface to the Vaisala WMT50 ultrasonic anemometer

The ultrasonic anemometer uses the time of flight of an ultrasonic pulse to determine both the speed and direction of the wind. There are no moving parts. The circuitry inside the anemometer takes care of the calculations and the formatting of the data for digital communications. We preconfigure the WMT50 to use its RS485 interface at 1200 baud, and we have an RS485 receiver based on the LT1785 chip at the OWL2pe end. It is further configured to deliver its data via in a simple format after it receives a command from the OWL2pe.

```
0R0Dm=xxxSm=xxx.xTh=xxx.xCRLF
```

Where Dm=xxx is the tag for direction, xxx is 0 to 359 degrees, Sm=xxx.x is the tag for speed, xxx.x in miles per hour, and Th=sxx.x is the tag for temperature, sxx.x in Celsius, signed. Here is the OWL2pe routine that asks for and receives the data.

```
wmt50_acquire:
 de485 CON 11   ' rs485 direction pin
 dt485 CON 10   ' rs485 data pin
 wbaud CON 813  ' 1200 baud
 HIGH de485
 SEROUT DT485,wbaud,["0R0!"]   ' command to WMT50 to send data
 LOW de485
 SERIN dt485,wbaud,1000,wmt50_timeout,[WAIT("0R0"),WAIT("Dm="),DEC ww,WAIT("Sm="),DEC wy1,DEC wy0,WAIT("Th="),SDEC wz1,DEC wz0]
'  DEBUG DEC ww,TAB,DEC wy1,".",DEC1 wy0,TAB,DEC wz1,".",DEC1 wz0,CR
 HIGH de485 : HIGH dt485  ' leave RS485 as driver asserted
  wy = wy1 * 10 + wy0       ' mph * 10    coding is wy1 is units, wy0 is tenths
  wz = wz1 * 10 + (-wz1.BIT7 ^ wz0) + wz1.BIT7      ' negative t sign extend
 ' returns with ww=direction degrees, wy=speed mph, wz=temperature Celsius
wmt50_continue:
 RETURN
```

The wind direction returned is not reliable when the wind speed is less than 1mph, so accumulations of direction are made only when the speed is greater than 1 mph.

Wind direction vane top

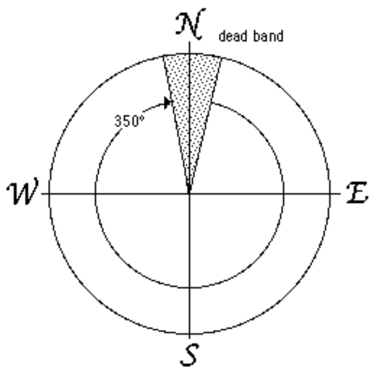
There are two ways we connect the wind vane:

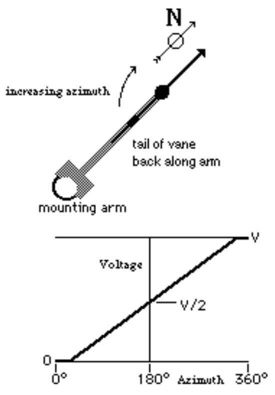
Wind vane pot connects between 4.096 volts reference output and common, and the wiper goes into one of the OWL2c input channels. The potentiometer produces a voltage between 0 and 4.096 that varies linearly with azimuth. This configuration requires three conductors in a cable from the logger out to the anemometer.

Wind vane pot is configured as a variable resistor, and this resistance is transmitted over 2 wires back to the data logger.   At the data logger, the variable resistance is in series with a fixed resistor to the 4.096 volt excitation, and the junction of the two is connectecd to the OWL2pe input channel.   The OWL2pe computes the resistance from the input voltage, and from that the azimuth.

The figure shows the recommended mounting, with the wind vane arm pointing out from the mast. There are often restraints on the mounting and considerations of the prevailing winds to take into consideration when deciding which way to point the arm.   The OWL2pe has provision to enter the direction that the mounting arm is actually pointed, and the OWL2pe compensates properly to return the correct azimuth. The Davis anemometers, as supplied from the factory, have the crossover point when the tail of the vane is back along the arm and the tip of the vane points out from the arm.   So that direction is nominally North.

The potentiometer in the wind vane is free to move through 360 degrees, but there is a dead band of about 10 degrees at the crossover point in the potentiometer travel. As shown in the graph, the output signal is zero volts for 5 degrees before the signal starts its linear rise, and the signal is maximum voltage through 5 degrees at the other extreme of the travel. The dead band varies some from unit to unit. If you are looking for the greatest possible accuracy, then you will need to find out exactly what the dead band is. You can also find out exactly what the dead band is by cutting a hole and a slit in a sheet of polar graph paper and putting it around the vane, and taking readings of w1 as a function of position of the vane in degrees. If you do not need great accuracy (5%) then simply use the typical value, 10 for the dead band calibration factor.





.

The following formula in the OWL2, BASIC Stamp, converts the reading from millivolts to azimuth:

```
WDead   con 10                       ' dead band in degrees
WDead2  con WDead/2                  ' half the dead band in degrees
WDfs    con 360-WDead                ' full scale span
gosub ADread                         'result is 0-5120 millivolts
azimuth = result */ WDfs / 20 + WDead2
```

The formula for azimuth produces in readings that go from 5 degrees to 355 degrees. The reading in the dead band is 5 degrees from zero to 5 degrees azimuth, and 355 degrees for 355 to 360 degrees azimuth.

{The */ operator is described in the document on math operations.}

```
      ' wind direction
      READ windAZcal0,Word wz  ' value in degrees that must be added to reading to give true direction
      ww=ww + wz +360//360  ' corrected azimuth
      IF wy>10 THEN PUT degWindAZ,Word ww  ' direction update only if windspeed > 1mph
      GET degWindAZ,Word wx    ' most recent value even if not updated
      GOSUB showdec0
```

If you only need 16 compass points, then you can use a simpler formula. Wind vanes often don't require much accuracy, and in such cases, 16 compass points may suffice.

```
    compass = result + 160 // 5120 / 320
```

16 compass points come out as 320 millivolts per compass point. The "+160" offsets the result, so that compass point zero comes out at 22.5 degrees on each side of north, and so on, the compass points are centered on the 16 cardinal directions.

When you put the vane up on a pole, you may have to orient it with the dead band pointing in a different direction. Then you only need to enter that direction in the calculation as an offset::

```
WDead   con 10                       ' dead band in degrees
WDead2  con WDead/2                  ' half the dead band in degrees
WDfs    con 360-WDead                ' full scale span
WDoff   con 105                      ' offset=105 degrees (example)
```

```
gosub ADread                         'result is 0-5120 millivolts
azimuth = result */ WDfs / 20 + WDoff +WDead/2 // 360
```

In this example, the azimuth goes from 110 degrees, smoothly through 360 to zero at North azimuth, and then back to 100 degrees at the other side of the dead band, and jumps back to 110 degrees at the midpoint of the dead band.

Here is a routine for averaging the aneometer direction component. Note that this has to be done using the sine and cosine components. Note also that a wind that blows half from one direction and half from its opposite will have an average direction of zero. Average direction is only of limited usefulness.

```
average_azimuth:
  ' at entry ww is current corrected azimuth 0 - 360
  ww = ww */ 182 ' convert to 0 to 256 brads
  wx = COS ww
  GET wdcos, Word wz
  wx = wx + wz
  PUT wdcos, Word wx
  wx = SIN ww
  GET wdsin, Word wz
  wx = wx + wz
  PUT wdsin, Word wx
  GET wdcos,Word ww
  GET samplesSP, Word wj
  wx = -wx.BIT15 ^ (ABS wx / wj) + wx.BIT15
  ww = -ww.BIT15 ^ (ABS ww / wj) + ww.BIT15
  wx =  wx ATN ww
  RETURN
```

The following algorithm keeps an accumulation of the compass points that are visited by the anemometer over a period of time. This is stored in word variable, so that each bit represents one compass point. At the end of an interval, you might find for example that this variable is %1100000000000001, which means the wind directions detected were N, NNE, and NNW.

```
GET windCompass,Word wx    ' get the current accumulation
IF wy>10 THEN
  wz = wz * 4 + 1485 // 1440 / 90 // 16    ' wz is 360 degrees as input, 0 to 15 output  with north =-11.25 to +11.25
  wx = DCD wz | wx   ' OR new direction into existing accumulation
  PUT windCompass,Word wx  ' only do update accumulation if speed is >1.0 mph
ENDIF
GOSUB showbin16    ' e.g. %1100000000000001
```

The following algorithm counts the number of ones as a measure of the total spread of wind directions:

```
' finds the spread
' counts the number of 1s in wx
countOnes:
N=0
DO WHILE wx
  wx = wx - (DCD (NCD wx - 1))  ' knock down one bit at a time
  N=N+1   ' count how many
  LOOP
  RETURN
```
