Introduction

The following sites were used as references:

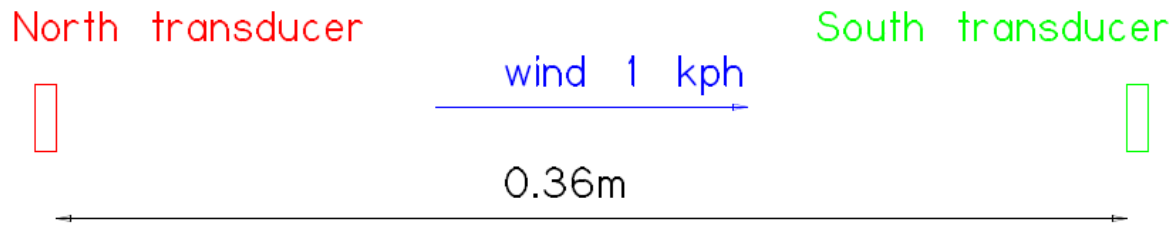
http://trekker.customer.netspace.net.au/wind.htm
http://www.circuitcellar.com/library/print/0106/Cyliax-186/index.htm
http://www.technik.ba-ravensburg.de/~lau/ultrasonic-anemometer.html

Separation of the transducers varies from 0.25 m to 0.36 m. The longer the path the easier it will be to determine the wind speed (larger tof changes).
I chose 0.36 m as being a reasonable physical size for the anemometer.

The transducers operate at 40,000 Hz which gives a pulse period of 25.0 micro sec.

The requirements of the anemometer:

I set the required resolution to be 1 kph (=0.277m/s).
At 22.0C the speed of sound C = 331.5 + (0.6 * 22) = 344.7 m/s.



The time of flight with no wind tof = 0.36 / 344.7 = 1044.3 micro sec.

tof from N to S tns = 0.36/(344.7 + 0.277) = 1043.5 micro sec.
tof from S to N tsn = 0.36/(344.7 - 0.277) = 1045.2 micro sec.

So 1 kph will give a difference of 0.8 or 0.9 micro sec in the tof.

We need a method of giving a resolution of at least 1 micro sec in the tof.
That is we must measure a time period of 1ms with an accuracy of one part in a thousand.
To do this we must maximize the signal to noise ratio. The normal electronic noise must be minimized by good circuit design.
A significant source of noise is from the wind itself. Any turbulence in the air movement will appear as "noise" in the tof measurements. We can ensure that the sensors are high up and clear from any obstructions. This helps but the wind is always turbulent to some extent.

Cross correlation:

http://www.imeko.org/publications/wc-2006/PWC-2006-TC4-028u.pdf

From this site we can see that the resolution is set by the sampling rate.
We would need a sampling rate of one every 1 microsecond. This would require an external high speed ADC.
The complete set of samples need to be in RAM.
We would only need an 8 bit ADC. If we use 600 pulses at 3 separate frequencies the memory requirement is to have 600 * 25 = 15000 bytes in RAM.

This method is very robust with low signal to noise ratios.

Thresholding:

This is the simplest method and I incorporate it in this design. The tof is measured when the amplitude of the received pulse exceeds a fixed amplitude.

In theory the error of this measurement can approach zero if you average an infinite number of samples.

The wind gust peak is usually averaged over 3 seconds(historical from the time constant of cup anemometers).

A sample run will take 3 second for the full north-south & east-west routine.
Each sample takes 2.4 ms. There will be 4 sets of samples so the maximum number of samples for each set = (3/4) / .0024 = 312.
I use 300 samples for each to allow for other calculations.

This method responds badly to low signal to noise ratios.

Phase shift:

In this method which I use we find the phase shift of the received signal compared with the transmitted signal.

The phase shift is proportional to the distance traveled.
One wavelength will give 360 deg phase difference.
As one wavelength is 25 micro sec we have a very fine measurement of time available.

I'm using Timer1 as a high speed counter and it counts 400 for each 25 micro sec.
With a resolution of 1 count the best possible time increment = 25/400 = 0.0625 micro sec.
The problem with this method is that the number of the wavelength is ambiguous.

This method is very robust with low signal to noise ratios.

Principle of operation:

I use the phase shift method to give the precise time measurement.

The less accurate threshold tof is used to determine the number of the wavelength in which we are measuring the phase shift.
With 300 samples we can determine accurately the number of the wavelength(multiples of 25 micro sec) .
So we get the the number of the 25 micro sec periods plus we can estimate to within 1 micro sec the phase shift of a partial period.

We can find the wind velocity from the equation:

Vns = d/2 *(1/tns - 1/tsn)

This requires that we have each tof as a discrete number.
This we do not have. We cannot use this equation.

What we have is a time difference of multiples of 25 micro sec.
We can solve for the magnitude of the wind velocity when we have the time difference:

V = SQRT((d/t)*(d/t) + C*C) - d/t

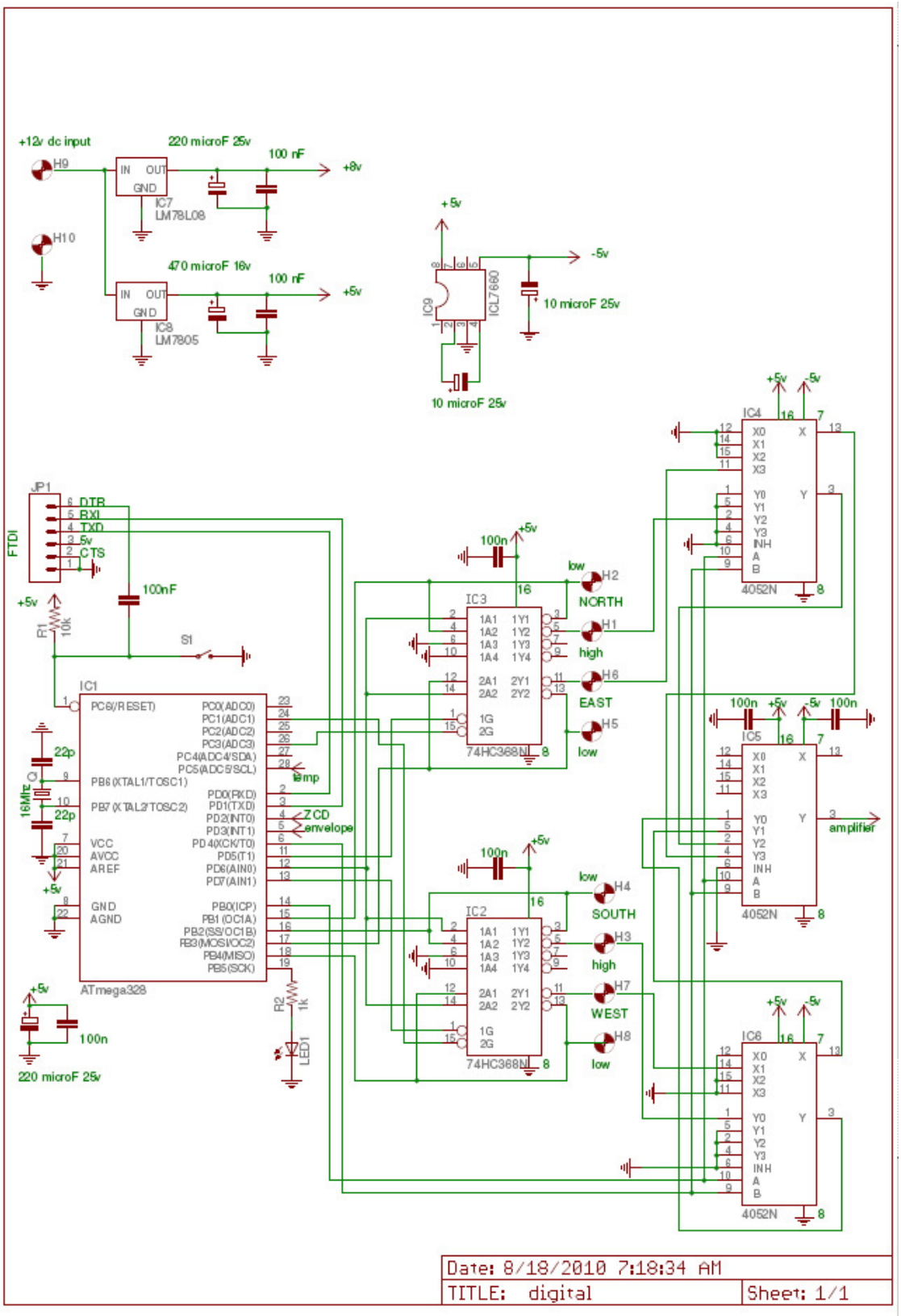
Where t = largest tof - smallest tof.
This is what we have with each wave length = 25 micro sec.

The wind velocity that gives a 25 micro sec difference at 22.0C(C= 344.7 m/s)
= 4.124 m/s = 14.85 kph.

We then find our wind velocity by knowing the number of 14.85 kph steps we have plus the part of 14.85 kph given by the phase shift.

There is one flaw in that the velocity step is dependant on ambient temperature.
The error over -10C to 50C is about +- 10%.
This is too large to have as our starting error so I have an ambient temperature sensor which is used to determine the wind step per wavelength.

Digital circuit


+12v dc input
H9
H10
220 microF 25v
100 nF
IC7
LM78L08
+8v
470 microF 16v
100 nF
IC8
LM7805
+5v
+5v
IC9
ICL7660
-5v
10 microF 25v
10 microF 25v
JP1
FTDI
DTR
RXI
TXD
5v
CTS
100nF
+5v
R1
10k
S1
IC1
PC6(/RESET)
PC0(ADC0)
PC1(ADC1)
PC2(ADC2)
PC3(ADC3)
PC4(ADC4/SDA)
PC5(ADC5/SCL)
temp
PB6(XTAL1/TOSC1)
PD0(RXD)
PD1(TXD)
PD2(INT0)
ZCD
envelope
PB7(XTAL2/TOSC2)
PD3(INT1)
PD4(XCK/T0)
PD5(T1)
PD6(AIN0)
PD7(AIN1)
VCC
AVCC
AREF
GND
AGND
PB0(ICP)
PB1(OC1A)
PB2(SS/OC1B)
PB3(MOSI/OC2)
PB4(MISO)
PB5(SCK)
ATmega328
22p
16MHz
22p
+5v
+5v
100n
220 microF 25v
R2
1k
LED1
100n
+5v
IC3
74HC368N
low
H2
NORTH
high
H1
EAST
H6
low
H5
IC2
74HC368N
low
H4
SOUTH
high
H3
WEST
H7
low
H8
IC4
4052N
IC5
4052N
amplifier
IC6
4052N
Date: 8/18/2010 7:18:34 AM
TITLE: digital
Sheet: 1/1

I have allocated a ATmega328 for the circuit(exact copy of an Arduino 2009) . It is fully occupied with the program to find wind speed and direction.

I use all three timers see:

http://www.atmel.com/dyn/resources/prod_documents/DOC2505.PDF

Timer0(Arduino millis() and PWM 5 & 6) is used as an exact frequency source for the ultrasonic pulses. I took every step to improve the S/N ratio. Fine changes in pulse frequency made a significant difference.
Driving the sensors directly using:

```
void pulsenorth()
{
 //PORTD is at 0x0b pin 6 controls NORTH HIGH  pin 7 controls NORTH LOW
 asm volatile ("\n\t"
            "ldi r25 , 16 \n\t"  //set pulse counter to 16
            "pulse1%=:" "\n\t"   //branch back here till 16 pulses sent

            "sbi %0 , 6  \n\t"  //turn on NORTH HIGH
            "cbi %0 , 7  \n\t" //turn off NORTH LOW

            "ldi r24 , 64 \n\t"  //set delay counter to 64
            "delay1%=:" "\n\t"   //branch back here gives 12.5 micro secs
            "dec r24      \n\t" // we have count one
            "brne delay1%=" "\n\t"  //stay in loop until set delay

            "cbi %0 , 6  \n\t"  //turn off NORTH HIGH
            "sbi %0 , 7  \n\t" //turn on NORTH LOW

            "ldi r24 , 64 \n\t"  //set delay counter to 64
            "delay2%=:" "\n\t"   //branch back here gives 12.5 micro secs
            "dec r24      \n\t" // we have count one
            "brne delay2%=" "\n\t"  //stay in loop until set delay

            "dec r25      \n\t" // we have done one
            "brne pulse1%=" "\n\t"  //stay in loop until reached 16 pulses

            "cbi %0 , 7  \n\t" //turn off NORTH LOW discharge 1.8 nF sensor


            ::"I" (_SFR_IO_ADDR(PORTD))  //PORTD used for NORTH 0x0b
            :"r24" , "r25");  //we have clobbered r24 and r25
}//end of pulse north
```

This is a C stub function see:

http://www.nongnu.org/avr-libc/user-manual/inline__asm.html

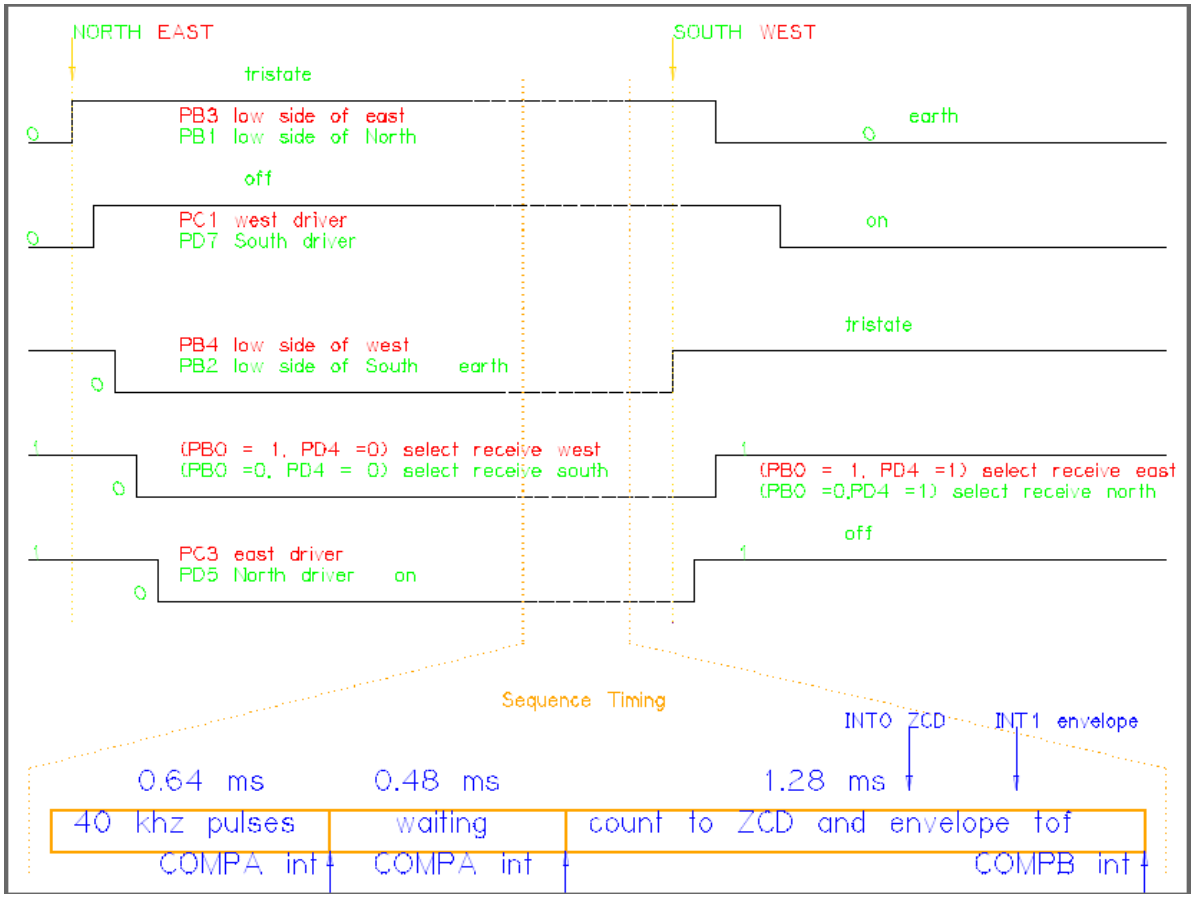
This does not give a fine enough control of the pulse frequency.

It means that I had to use 2 74HC368 drivers, but it is not to extravagant to get the best S/N ratio.

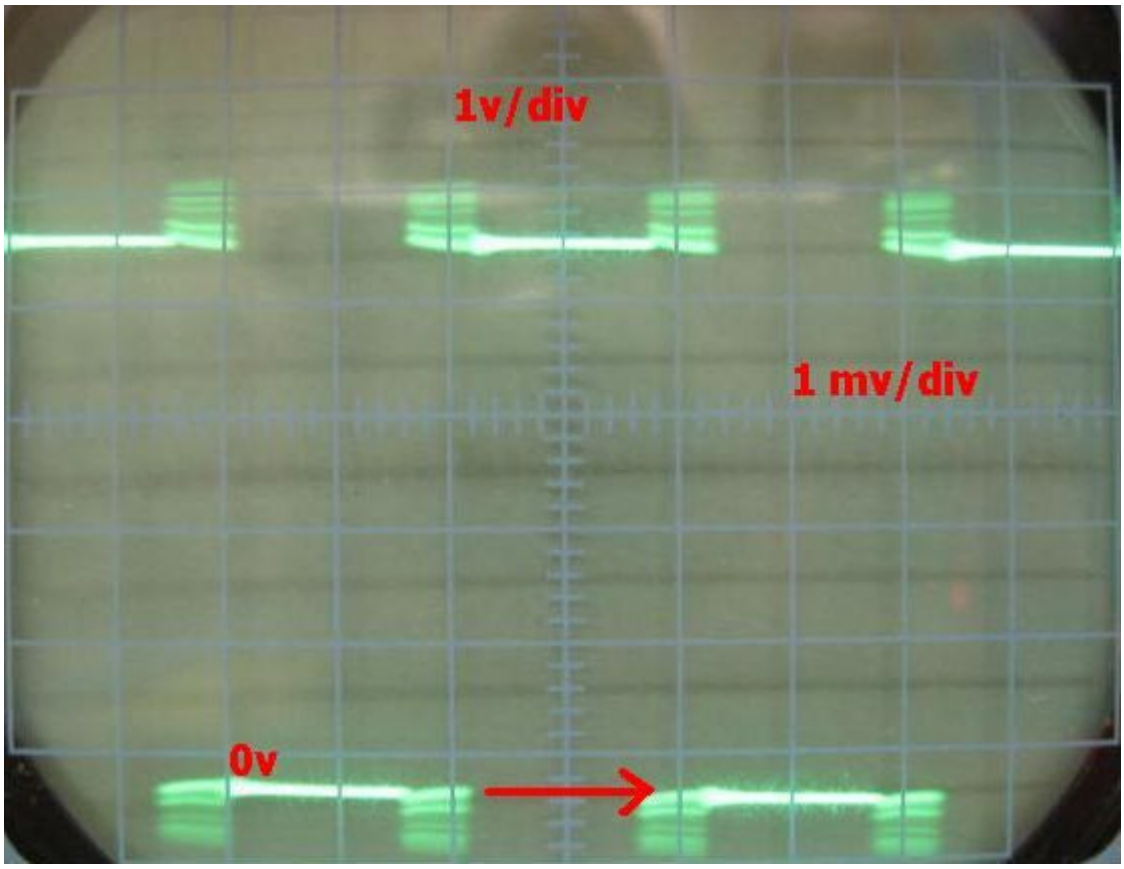
Timer1(Arduino PWM 9 and 10) is used as a high speed counter. The clock is not pre-scaled and it counts 400 for each 25 micro sec.
It is used for both the phase detector and threshold detector counter.
The phase detector is set to trigger first and INT0 is used to store that count.
The counter continues on and then the threshold detector triggers INT1 which is used to store that count.

Timer2(Arduino PWM 3 and 11) is used as the system timer. We have 3 steps which use Timer2 interrupts COMPA and COMPB.

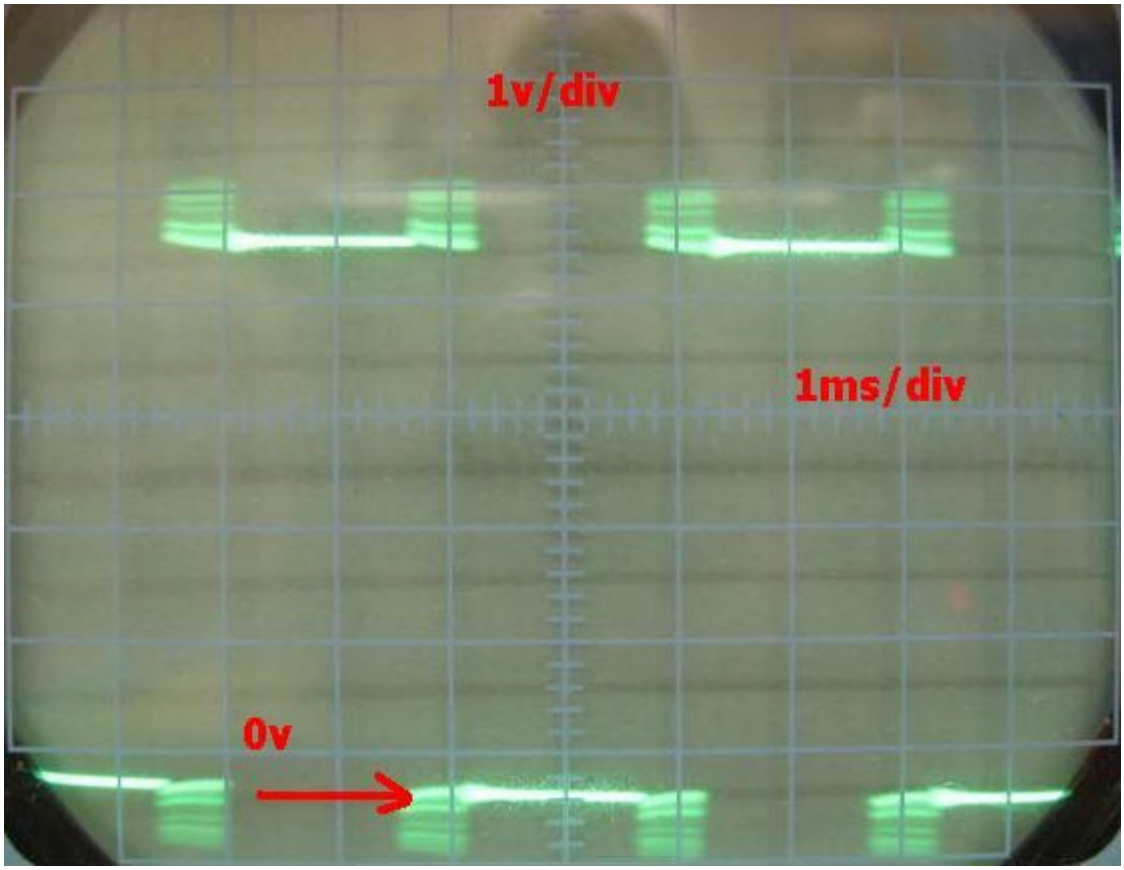
1: Send 40 khz pulse for 0.64 ms.
2: Dead zone of 0.48 ms as the system settles.
3: Active waiting of 1.28 ms for INT0 and INT1.



The connection and timing of all the control lines are shown in the above diagram.



CRO display at North high, East high,South high or West high with program CRO.



CRO display at North low, East low ,South low or West low with CRP program.

I use 2 stages of 4052 multiplexers. The crosstalk through one 4052 is -40 db which is a voltage gain of 1/100. As the adjacent channel has 5 v pulses the direct cross talk would be 50 mv. This is way bigger than the received signal(5mv) so the extra stage is necessary to ensure a very clean received pulse.

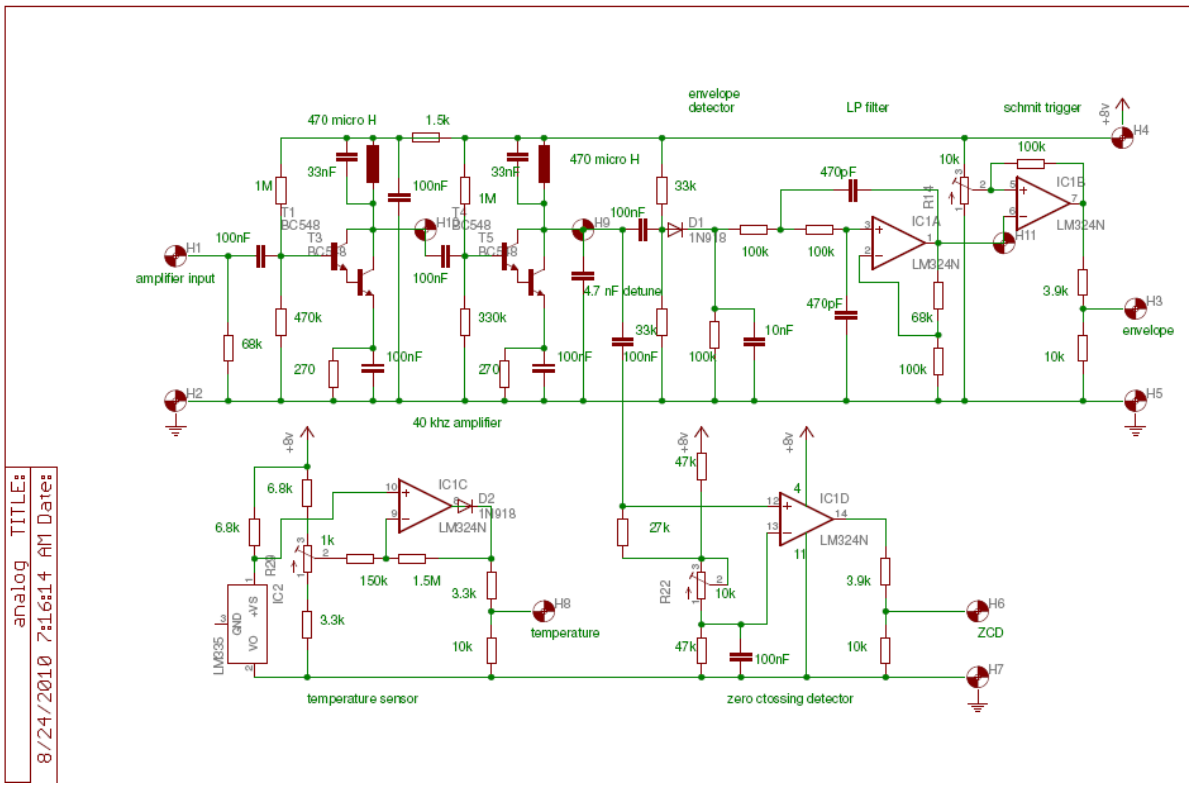
The transducers I chose were the T/R40-16B, a generic brand used for such purposes as a proximity detector on cars. They have of course have to be weatherproof.

http://www.datasheetarchive.com/pdf-datasheets/Datasheets-9/DSA-175857.html

They are cheap(AUD 5) and have virtually the same specs as some much more expensive name brands.

The important specs are for transmitting:
Sound pressure > 105 db
and for receiving:
sensitivity > - 74db

Analog circuit

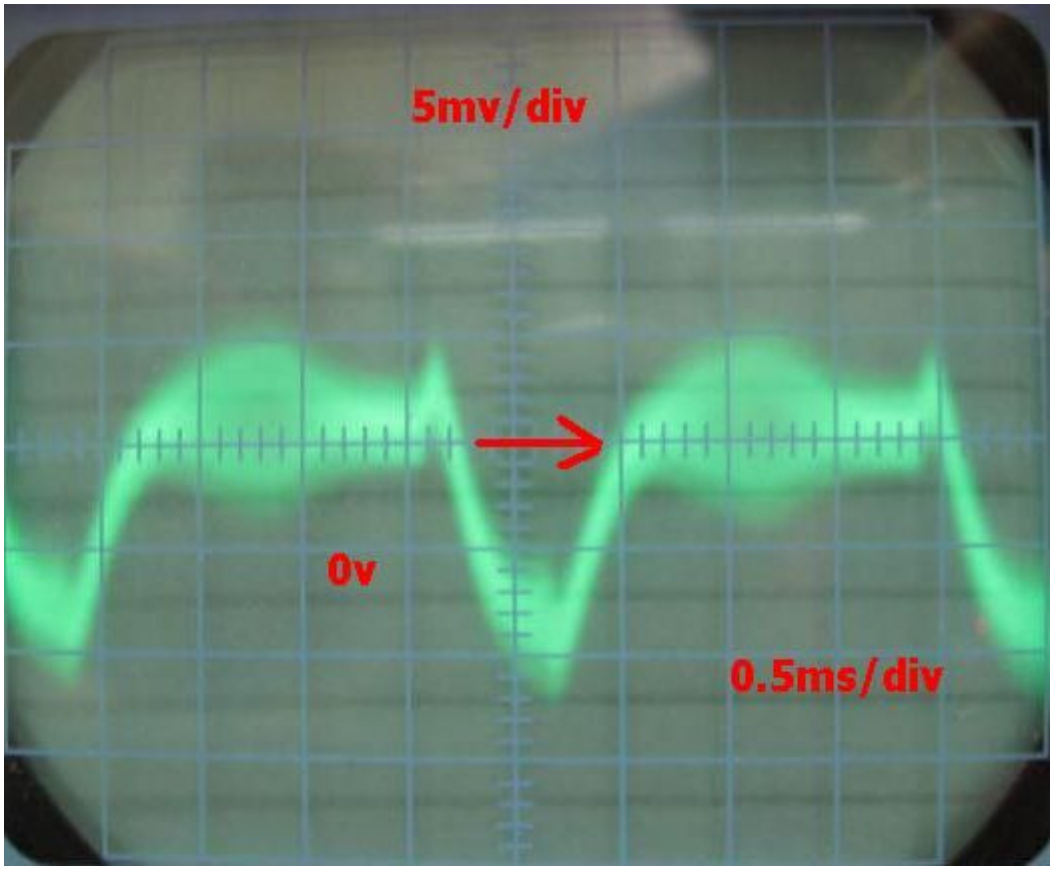


Amplifier circuit:

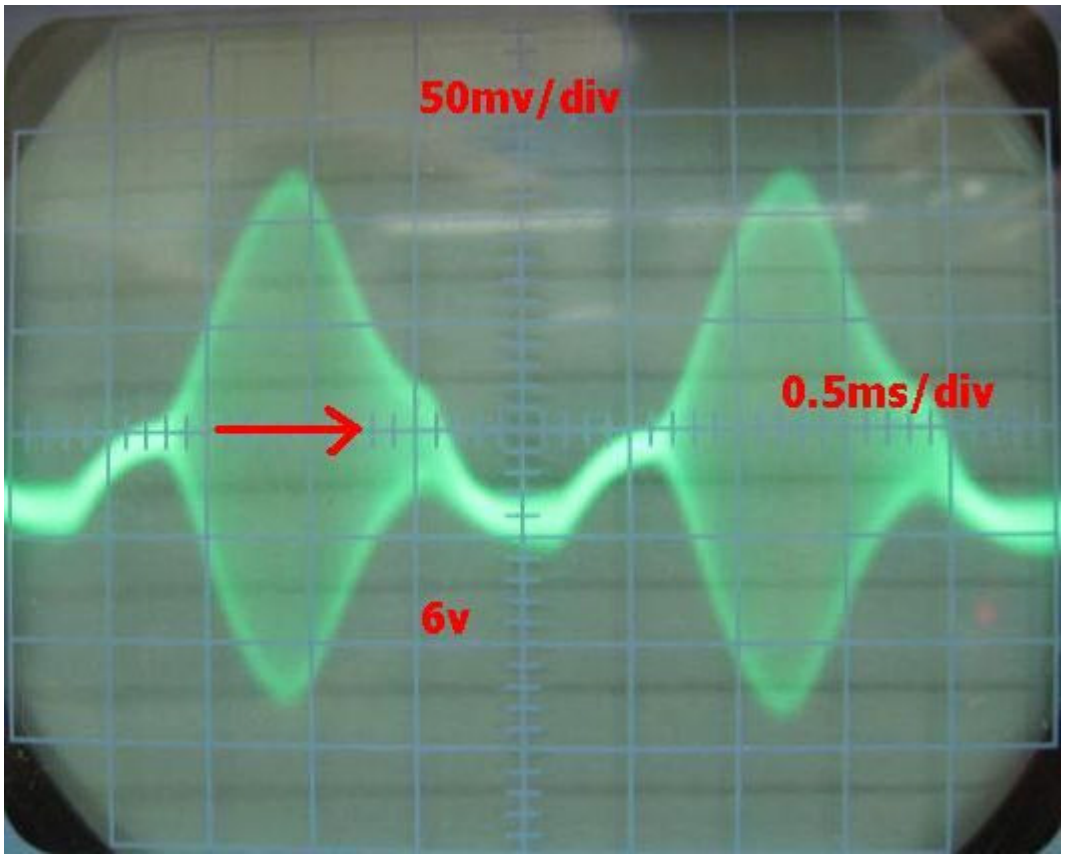
The noise figure of an amplifier is set by the first stage. To get the best noise figure available I used a bjt tuned amplifier for the first stage. The band width is set by the 472 mico H inductor. I use a standard rf choke with a resistance of about 10 ohms. This gives a Q of about 10 and a bandwidth of about 4 khz. The stage voltage gain is about 50.

The second stage could have been a op amp but for pragmatic reasons I copied the first stage. The maximum voltage gain possible for a stable fixed frequency amplifier is about 1000. For this reason it was necessary to detune the 2nd stage with the 4.7 nF capacitor. This sets the second stage voltage gain to about 10.
A simple check on the amplifier operation is to open the input and it will oscillate with full output voltage swing(+-8v) at 40 khz.

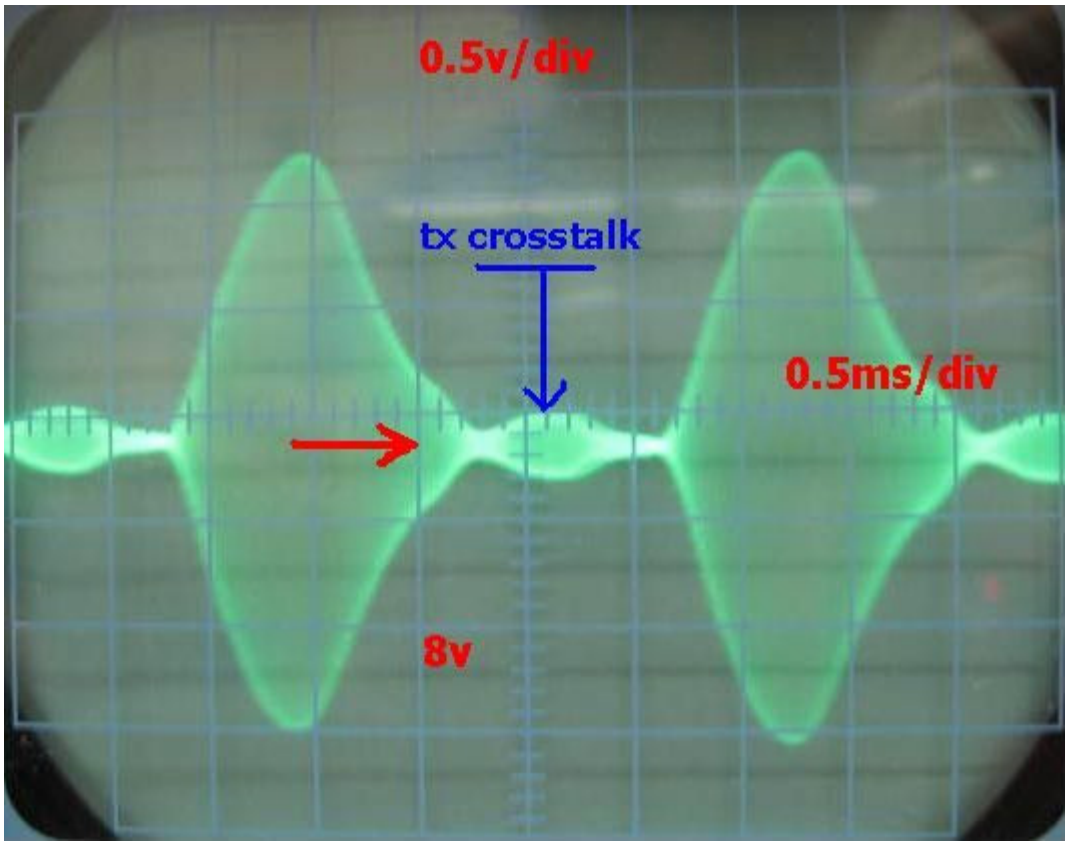
The two stages together give a BW of about 3 khz.
The darlington pairs are used to increase the input impedance so that the transducer and the first tuned circuit are not loaded down.



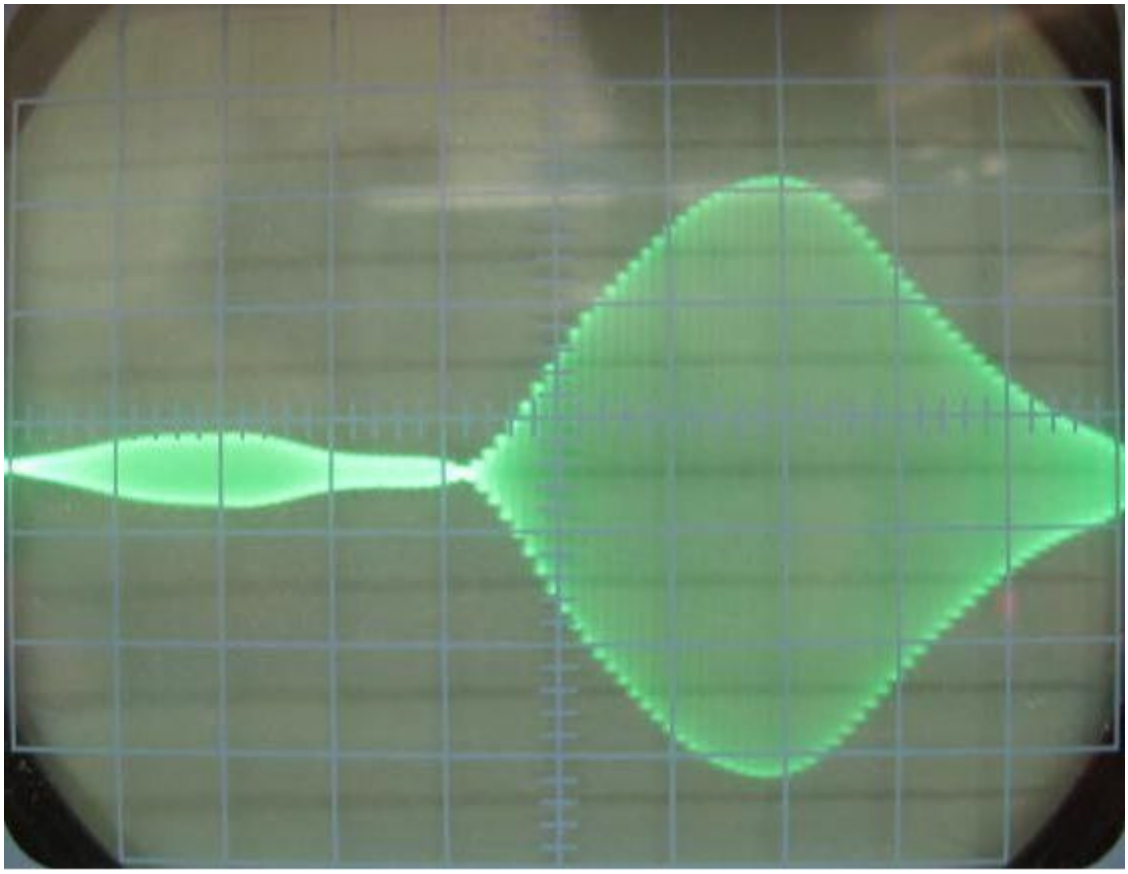
Amplifier input with example program CRO



Output from 1st stage with example program CRO.



Output from 2nd stage of the amplifier with example program CRO.



Expanded output from 2nd stage of the amplifier with example program CRO.

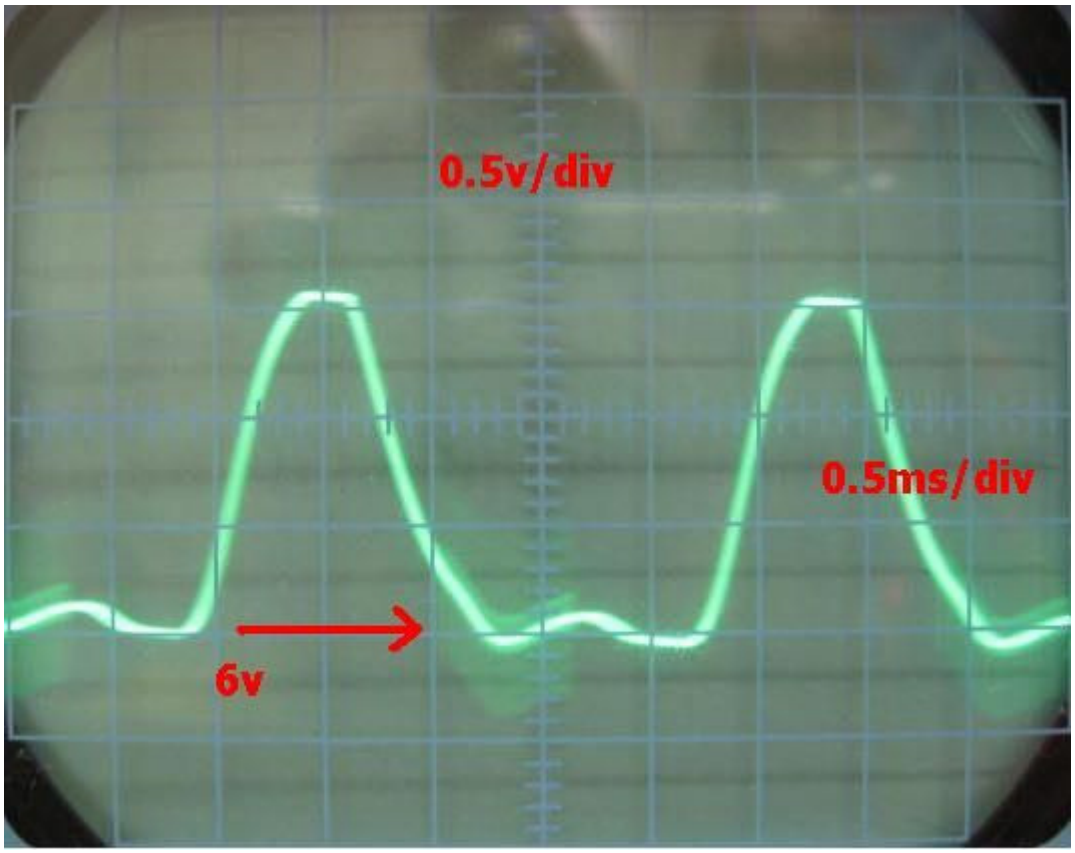
Envelope detector:

I use a schmitt trigger as a threshold detector. This is to ensure we don't have multiple transitions at the threshold.

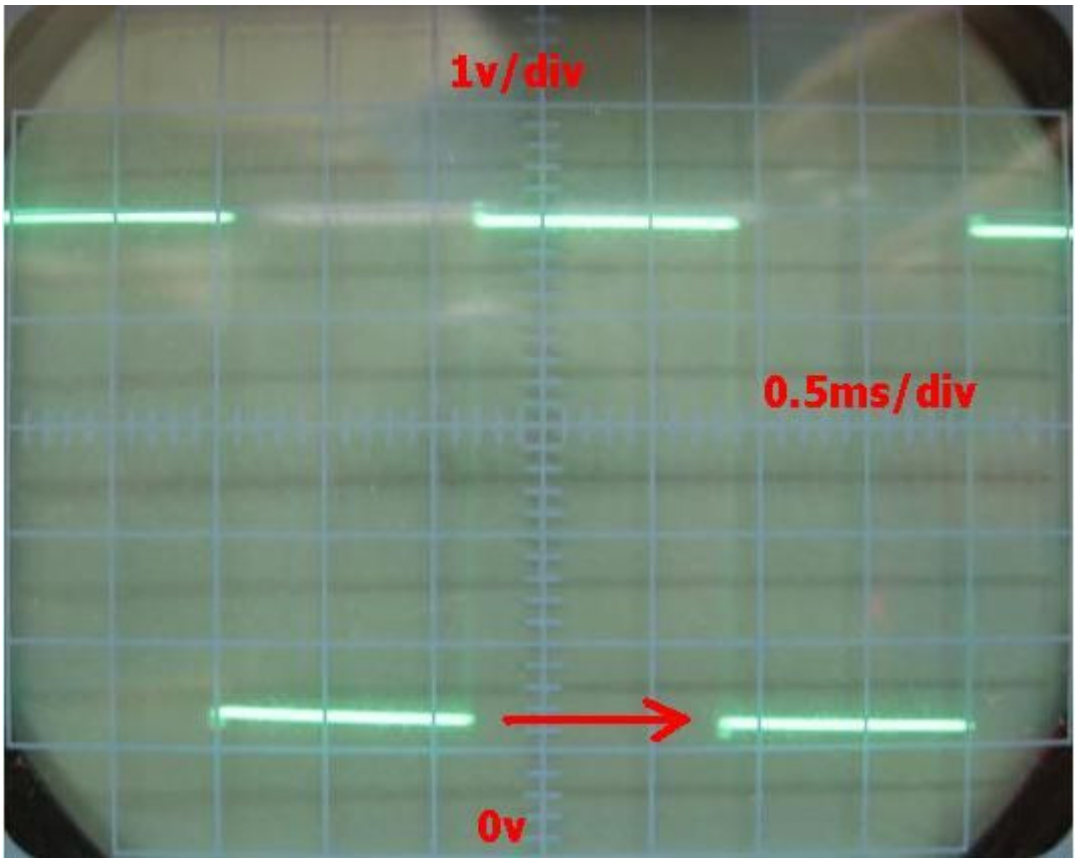
To get a smooth envelope for the schmitt trigger we need to have an envelope detector followed by a low pass filter.

The envelope detector is identical in operation to that used to detect AM signals in an AM receiver.

The low pass filter is a voltage-controlled voltage-source (VCVS) circuit. It implements a two pole Butterworth with a corner frequency set by the 100k and 470 pF to be 3500 khz.



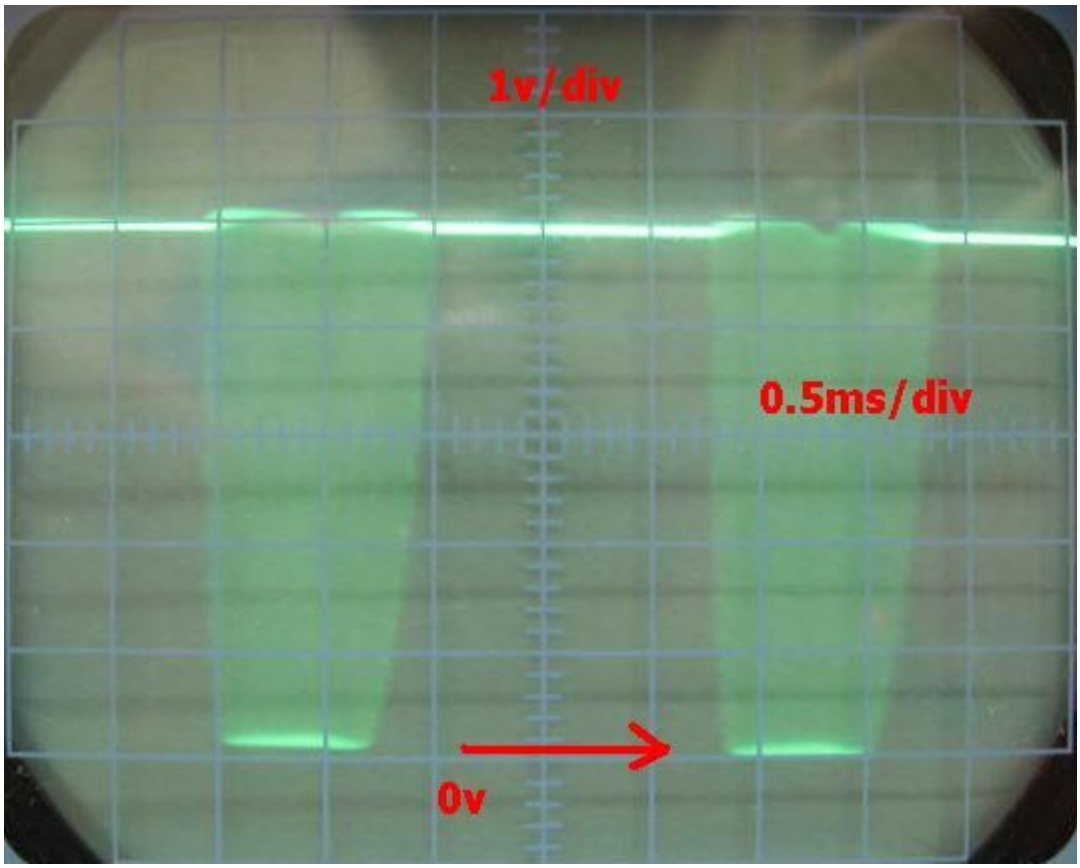
Output from the LPF with example program CRO.



Output from the envelop detector with example program CRO.

Zero crossing detector:

This circuit is a comparator with the an adjustable threshold set by the variable resistor.



Output from the zero crossing detector with example program CRO.

Temperature sensor:

Here we use a simple sensor based on a LM335 see:

http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1259616399

Program operation

interrupts:

All the interrupt service routines are are detailed in:

http://www.nongnu.org/avr-libc/user-manual/group__avr__interrupts.html

operation sequence:

1. Setup to transmit from North to South and get 300 samples.

```
NORTH();
for(counter = 0; counter < 300; counter++)
{
sequence();  //transmit and receive pulse
NZCD[counter] = zcdtof;
if ( counter != 0)  //first one no good
Naverage += envelopetof;
}//end of fill the zcd samples
```

NORTH() sets up the control lines to transmit the 40 khz pulses from the north transducer. It also sets up to receive and amplify the signal from the south transducer.

sequence() controls the system timer timer 2 to transmit the pulses for 0.64 ms, rest for 0.48 ms and complete the sequence in 2.4 ms.

In the sequence subroutine we have:

```
inline void sequence(void)
{
 OCR0A = pulsefrequency; //set this constant to give best response
 TCCR0A = 0x42;  //toggle OC0A CTC mode count to OCR0A
 TCCR0B = 0x01;  //use prescale = /1 start counting pulse generator
 TCNT0 = 0x00;  //set start value of counter

 TIFR2 = 0x07; // clear all flags
 TIMSK2 = 0x04;  //enable interupts when count to OCR2B
 OCR2B = 40;  //40 gives interupt in 0.64 ms
 TCCR2A = 0x00;  //normal port operation counts to max 255
```

```
 TCCR2B = 0x06;  //use prescale  /256 start count
 TCNT2 = 0x00; //after .64 ms stop pulse generator and start tof count

 pulsedone = false; //wait for system timer
 pulsesent = false; //wait for system timer
 while( ! pulsedone)   //TIMER2 will set after 2.4 ms
 asm volatile("nop");
}//end of timing sequence
```

TIMER0 is used as the pulse generator. The first 4 lines set up the various TIMER0 registers. It produces a fixed pulse frequency set by the constant pulsefrequency.

The next 6 lines setup TIMER2 to give an interrupt in 0.64 ms using COMPB.
In the ISR which handles the interrupt when the count reaches OCR2B we have:

```
ISR(TIMER2_COMPB_vect)
{
  if( ! pulsesent)
  {
  OCR2B = 30;//wait 0.48 ms to start the tof counter
  TCCR0B = 0x00; //stop the pulse driver
  pulsesent = true; //the pulse has been sent
  TCNT2 = 0x00;
  }//end of sentpulse wait another .48 ms till start counter
  else
  {
  OCR2A = 80; //80 gives a total interupt in 2.4 ms
  TIMSK2 = 0x02;  //enable interupts when count to OCR2A
  TCNT2 = 0x00; //after 2.4 ms drop out of delay loop after pulse received
  EIFR = 0x03;  //clear INT0 and INT1 flag they may have been previously set
  EIMSK = 0x03;  //enable interupts on INT0 and INT1
  TCNT1 = 0x00;  //start the tof count
  }//end of delay after sent pulse
}//end of interupt when OCR2B count reached -pulse sent
```

The first time we get this interrupt pulsesent is false so that we execute the code in the first block.
The count OCR2B is reset to 30 which gives an interrupt in 0.48 ms.

The second time we now have pulsesent true so the second block is executed.
OCR2A is set to give a time delay of 1.28 ms.

The interrupts INT0 and INT1 are enabled and the counter TIMER1 is started at zero.

Before the interrupt caused by OCR2A we have firstly INT0:

```
ISR(INT0_vect)
{
 zcdtof = TCNT1; //save the counted tof value
 EIMSK &= ~0x01; //disable INT0 we have counted this tof
}// INT0 service routine for zero crossing detector
```

Then we will have INT1:

```
ISR(INT1_vect)
{
 envelopetof = TCNT1; //save the counted tof value
 EIMSK = 0x00; //disable INT1 we have counted this tof
}// INT1 service routine for envelope detector
```

We now have two samples. The ZCD count is stored in an array because we need to access the individual tof's for each pulse.

Mean time back in the main program we are sitting at:

```
 while( ! pulsedone)   //TIMER2 will set after 2.4 ms
 asm volatile("nop");
```

When OCR2A reaches 80 we get an interrupt:

```
ISR(TIMER2_COMPA_vect)
{
 TCCR2B = 0x00; //stop the system timer
 TIMSK2 = 0x00;  //disable all interupts on system timer
 pulsedone = true; //we have finished sequence
}//end of interupt when OCR2A count reached -pulse received
```

When this occurs we drop out of the wait loop .

We run this sequence 300 times to fill the NZCD[counter] and find the total of the envelope tof's in Naverage.

2. Setup to transmit from South to North and get 300 samples.

The whole process is repeated as for step 1 until we have the SZCD[counter] filled and the envelope total in Saverage.

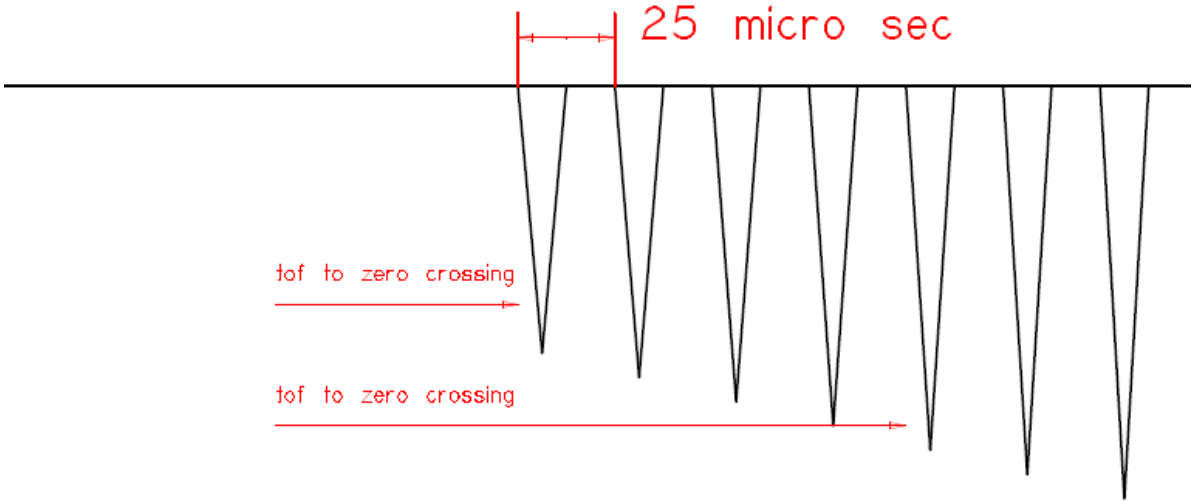
3. Firstly we need the phase shift. I use the phase difference between each pulse north and the matching pulse south.

diff = NZCD[counter] - SZCD[counter];

We encounter a problem called pulse shift:

Because of noise we may not be using the corresponding pulse.
We may be out by an integral number of pulse widths. Each is 25 micro sec or count of 400.

Note the number is not exactly 400 because of timing delays in the interrupt.
The constant zcdwidth needs to be tuned to find the exact value.



We need to incorporate this by:

```
int diff = 0;
long diffav = 0;
for(counter = 3; counter < 300; counter++) //first 3 are not good
{
diff = NZCD[counter] - SZCD[counter];

while ( diff < - 200) //a minus pulse slip so add a pulse
diff +=400; //there may be more than one pulse slip

while ( diff > 200) //a plus pulse slip so subtract a pulse
diff -=400; //there may be more than one pulse slip

diffav += diff;//sum the values

}//end of zcd samples
```

We get a number diffav which can be as large as 297 * 400 = 59400.
In practice the maximum is around 55000.
This is directly proportional to the phase shift between the north and south pulses.

Next we need to find the difference in tof's between the north and the south envelopes
For this we can just use the averages as we do not have any ambiguity with pulses.

Naverage = Naverage/299;
Saverage = Saverage/299;

We do run into another problem. Ultrasonic transducers are not symmetric for transmission and reception. The devices are not linear components and even worse the symmetry changes with temperature.

You can match them as north-south, east-west pairs. It's properly a good idea to do so. However we do need to have a constant which is used to match the pairs as closely as possible. This constant will need to change with time as the temperature changes.

I use:

int Northerror[20]; //The tof error for north-south transmit

We keep a running sample of the differences when the difference should be zero. This occurs with zero wind or at other zeros in the wind function(described later).

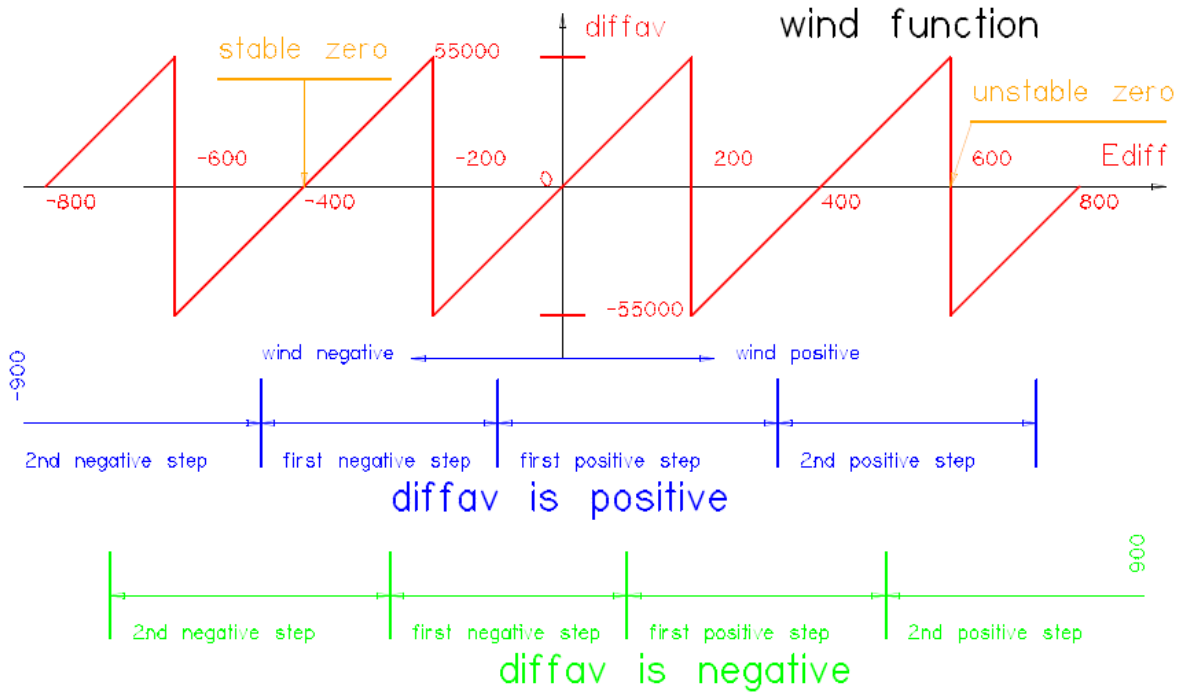
```
if ((diffav > -5000)  && (diffav < 5000)) //zero in zcd?
 {  //if a zero in ZCD the tof difference should be zero
 for( counter = 0; counter < 19; counter++ )
 Northerror[19-counter] = Northerror[18-counter]; //shift up one
 if (Naverage > Saverage) //is difference positive
 Northerror[0] = -((Naverage - Saverage)%400);  //put in new error term
 else  //difference is negative
 Northerror[0] = (Saverage - Naverage)%400;  //put in new error term
 }//end of we have a zero in the zcd
 int Nerror = 0; //the north-south error tof error value
 for( counter = 0; counter < 20; counter++ )
 Nerror += Northerror[counter];
 Nerror /=20;  //the average of the tof zero error

 Naverage += Nerror;  //compensate the tof value
```

We now have our matched envelope tof's. we can find the difference:

```
 int Ediff = Naverage - Saverage;
```

We get a number Ediff which can vary from zero with zero wind to +- 8* 400 with wind speed up to +-100kph.

If we now plot diffav vrs Ediff we get the wind function:



The shape of this function is determined by the method of finding diffav in the program. The zeros in the wind function are shown. The unstable zeros will never occur in practice.
The number will jump from max to min or visa versa. The wind is never steady in strength or direction and as long as occasionally we get a zero the envelope correction constant can be updated. This is the reason for using:

```
Northerror[0] = (Saverage - Naverage)%400;
```

Each step of 400 in the wind function correspond to a wind speed given by:

V = SQRT((d/t)*(d/t) + C*C) - d/t in m/s
where d = 0.36 m and t = 25 micro sec
so d/t = 14400.0 , (d/t)*(d/t) =207.36E6

so we get the step wind speed in kph:

```
int Ext = analogRead(tempinput);
float Exts =(Ext *60.0)/1024.0 - 10.0; //the temperature in centigrade
Exts = 331.5 + (0.6 * Exts);  //the speed of sound at the ambient temperature
float windstep = (sqrt((207.36E6 + Exts*Exts)) - 14400.0)*3.6;
//the wind step in kph at the ambient temperature
```

We can then find the actual wind using this code:

```
boolean function = false;
 if(diffav < 0) //we are on the negative side of wind function
 {
  function = true;
  diffav = - diffav;  //invert the wind function
  Ediff = - Ediff;  //and  Ediff
 }//end of negative side of wind function
 float wind = (windstep * diffav)/110000.0; //must have this fine value
 if(Ediff > -100)  //we have positive wind
 {
 int pstep = -1;
 while(Ediff >-100)
 {
  pstep +=1; //count each step
  Ediff -= 400;  //move one step
  if(pstep == 7)
  Ediff = -1;  //we have exceeded our range 100 kph max
 }//end of count number of steps positive
 wind += pstep * windstep; //this is final wind speed
 }//end of we are on the positive side
 else    //we have negative wind
{
int nstep = 0;
while(Ediff < -100)
{
 nstep += 1; //count each step
 Ediff += 400;  //move one step
 if(nstep == 8)
 Ediff = 1;  //we have exceeded our range 100 kph
}//end of count number of steps negative
 wind -= nstep * windstep; //this is final wind speed
}//end of we are on negative side
 if (function)
 wind = - wind;//the function has been inverted
```

We have used a count of 100 in Ediff as the dividing points.
As long as Ediff accuracy can be kept to +- 100 (+-6.25 micro secs) the wind speed will be as accurate as the diffav which is very precise and robust.

4. We now repeat steps 1,2 and 3 but this time for the East West pair of transducers.

I have used the same arrays and variables as with North South so that the code is almost a exact copy of that used for steps 1,2 and 3.

5. We now have the apparent wind in the north-south and the east-west direction.

We can now calculate the actual wind speed and direction:

I use the convention that the wind direction is given as the direction from which it blows.
North is taken as 0 deg.
The North-South and East-West pairs are opposite each other.
Then observe which wind direction causes which wind speed in the monitor program.

If we get a positive wind for North-South the closest sensor is north.
If we get a positive wind for West-East the closest sensor is West.

```
//N_Swind is +ve if wind blowing from the North. North is 0 deg on the compass
//W_Ewind is +ve if wind is blowing from the West. West is 270 deg on the compass.

int Awind = sqrt(N_Swind*N_Swind + W_Ewind*W_Ewind);

int Dwind =57.3*atan2(N_Swind,W_Ewind) + 270.0; //270 moves result into compass
Dwind %= 360; //we are only use 360 degree circle
```

I have converted the floating point to an integer for both wind speed and direction as this reflects the precision of the sensor.

Construction

I have use pvc pipe throughout. It is painted white for protection from UV and the heat of the sun.

The sensor arms are 15mm pvc pressure pipe. All standard fitting are used. We have a cross, 45 deg and 90 deg joints.

The cross is bolted to the top of a 100 mm drainage cap.

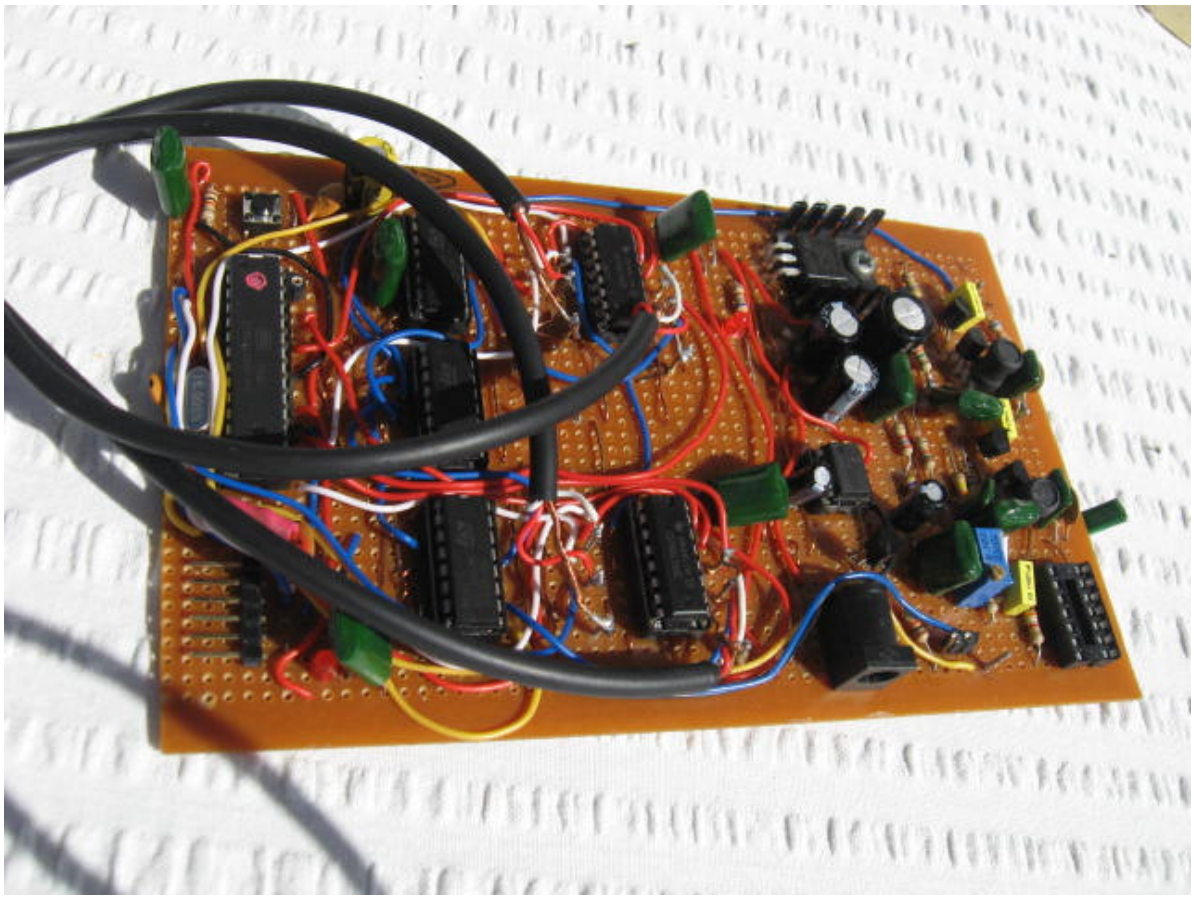
The body is 100 mm drainage pipe with a bottom end cap.
The sensors are mounted in the projecting pipes.
One can be seen here:

The bottom pipe mount is 32 mm drainage bolted to the bottom cap with a 32mm to 75mm flange.

The protective cover for the ambient temperature is a 15 mm joiner.
We have some holes in it to allow air circulation.

The vero board I used for the electronics fits neatly into the main body.

There are a couple of extra components(mods) mounted on a small vero board and connected to the open dil socket.



Calibration Programs

calibrateCRO:

Run this sketch to cycle through the four transmit/receive stages.
This will give the CRO traces shown previously.

calibrateTEMP:

Use this program to calibrate the temperature sensor.
Adjust the 1 k pot to give the known ambient temperature.
You should see:

```
external temperature = 24.10
external temperature = 24.16
external temperature = 24.16
external temperature = 24.10
external temperature = 24.10
external temperature = 24.04
external temperature = 24.10
external temperature = 24.10
external temperature = 24.16
external temperature = 24.10
```

calibrateZCD:

This sketch will find the average of each tof samples using the zero crossing detector.
Adjust the 10k pot to give about 3000.
You should see:

```
North = 3187 South = 3206.....East = 2998 West = 3016
North = 3189 South = 3202.....East = 2998 West = 3015
North = 3190 South = 3201.....East = 2946 West = 3008
North = 3189 South = 3207.....East = 2961 West = 3008
North = 3188 South = 3202.....East = 2991 West = 3009
North = 3192 South = 3200.....East = 2966 West = 3011
North = 3193 South = 3209.....East = 2994 West = 3010
North = 3192 South = 3209.....East = 2964 West = 3009
North = 3197 South = 3206.....East = 2992 West = 3009
North = 3190 South = 3201.....East = 2990 West = 3011
North = 3188 South = 3208.....East = 2995 West = 3009
North = 3190 South = 3207.....East = 2969 West = 3010
North = 3187 South = 3199.....East = 2995 West = 3011
North = 3186 South = 3196.....East = 2998 West = 3012
```

calibrateENVELOPE:

This sketch will find the average of each tof samples using the envelope detector.
Adjust the 10k pot to give about 4000.
You should see:

North = 4237 South = 4182 Difference = 55.....East = 3972 West = 3959 Difference = 13
North = 4238 South = 4186 Difference = 52.....East = 3961 West = 3934 Difference = 27
North = 4235 South = 4182 Difference = 53.....East = 3961 West = 3929 Difference = 32
North = 4228 South = 4181 Difference = 47.....East = 3974 West = 3936 Difference = 38
North = 4222 South = 4186 Difference = 36.....East = 3959 West = 3940 Difference = 19
North = 4235 South = 4181 Difference = 54.....East = 3969 West = 3959 Difference = 10
North = 4236 South = 4190 Difference = 46.....East = 3947 West = 3948 Difference = -1
North = 4233 South = 4183 Difference = 50.....East = 3949 West = 3959 Difference = -10
North = 4235 South = 4183 Difference = 52.....East = 3937 West = 3938 Difference = -1
North = 4239 South = 4174 Difference = 65.....East = 3946 West = 3927 Difference = 19
North = 4234 South = 4172 Difference = 62.....East = 3944 West = 3927 Difference = 17
North = 4224 South = 4176 Difference = 48.....East = 3940 West = 3942 Difference = -2
North = 4232 South = 4169 Difference = 63.....East = 3950 West = 3932 Difference = 18
North = 4234 South = 4174 Difference = 60.....East = 3949 West = 3903 Difference = 46
North = 4241 South = 4178 Difference = 63.....East = 3984 West = 3962 Difference = 22
North = 4227 South = 4176 Difference = 51.....East = 3964 West = 3949 Difference = 15

monitor:

This program sends all the required data you need to monitor the operation of the sensor.
I ran it for a few weeks before changing to the operate program.
you should see:

T= 24.69 z= 336 N= 4210 S= 4144 e= -49 w= 0.05....z= 1352 E= 3977 W= 4047 e= 58 w= 0.
18.....Aw= 0 Dw= 283
T= 24.75 z= 621 N= 4198 S= 4151 e= -49 w= 0.08....z= 194 E= 3965 W= 4020 e= 58 w= 0.
03.....Aw= 0 Dw= 342
T= 24.69 z= 193 N= 4187 S= 4144 e= -48 w= 0.03....z= -302 E= 3955 W= 4022 e= 58 w= -0.
04.....Aw= 0 Dw= 57
T= 24.75 z= 689 N= 4190 S= 4138 e= -48 w= 0.09....z= 2062 E= 3945 W= 4053 e= 61 w= 0.
28.....Aw= 0 Dw= 288
T= 24.75 z= 344 N= 4189 S= 4138 e= -48 w= 0.05....z= 1680 E= 3988 W= 4032 e= 61 w= 0.
23.....Aw= 0 Dw= 281
T= 24.69 z= 650 N= 4198 S= 4140 e= -49 w= 0.09....z= 2434 E= 4004 W= 4040 e= 60 w= 0.
33.....Aw= 0 Dw= 284
T= 24.80 z= 521 N= 4193 S= 4154 e= -48 w= 0.07....z= 2487 E= 3992 W= 4048 e= 61 w= 0.
34.....Aw= 0 Dw= 281
T= 24.75 z= 744 N= 4200 S= 4153 e= -48 w= 0.10....z= 1765 E= 3993 W= 4036 e= 60 w= 0.
24.....Aw= 0 Dw= 292
T= 24.75 z= 731 N= 4200 S= 4150 e= -48 w= 0.10....z= 1735 E= 3973 W= 4033 e= 61 w= 0.
24.....Aw= 0 Dw= 292
T= 24.75 z= 755 N= 4199 S= 4141 e= -48 w= 0.10....z= 591 E= 3976 W= 3986 e= 59 w= 0.
08.....Aw= 0 Dw= 321

T= 24.75 z= 694 N= 4182 S= 4133 e= -49 w= 0.09....z= 592 E= 3956 W= 4009 e= 59 w= 0.
08.....Aw= 0 Dw= 319
T= 24.69 z= 587 N= 4184 S= 4129 e= -49 w= 0.08....z= 2747 E= 3997 W= 4058 e= 60 w= 0.
37.....Aw= 0 Dw= 282
T= 24.69 z= 724 N= 4186 S= 4133 e= -49 w= 0.10....z= 1915 E= 4001 W= 4037 e= 59 w= 0.
26.....Aw= 0 Dw= 290
T= 24.75 z= 614 N= 4179 S= 4132 e= -49 w= 0.08....z= 1399 E= 4007 W= 4039 e= 58 w= 0.
19.....Aw= 0 Dw= 293
T= 24.75 z= 667 N= 4189 S= 4149 e= -48 w= 0.09....z= 2060 E= 3999 W= 4047 e= 58 w= 0.
28.....Aw= 0 Dw= 287
T= 24.75 z= 1065 N= 4211 S= 4159 e= -48 w= 0.15....z= 1775 E= 4005 W= 4028 e= 56 w= 0.
24.....Aw= 0 Dw= 300
T= 24.69 z= 1016 N= 4208 S= 4154 e= -49 w= 0.14....z= 1672 E= 3986 W= 4030 e= 53 w= 0.
23.....Aw= 0 Dw= 301

operate:

This program sends two integers. The wind speed and the wind direction.
The timing is about 3 seconds per each transmission.