A PICAXE-based Audio Frequency Decoder..

.... with the emphasis on measuring the actual audio frequency.

It does NOT decode signal types like FSK, PSK, CW....

February 2010

If you have visited other pages on the VK4ADC web site, you may have noticed a few transverter (23cm) and PLL project pages. One comment that you will have seen repeated on them and on my Field Day reports has to do with frequency drift - the local oscillator in the receiver/transceiver or transverter changing frequency with time and temperature. It is relatively easy to watch the frequency change by the use of software like Digipan (http://www.digipan.net/) or Spectran (http://www.sdrham.com/spectran.html) to observe the tone output of a receiver tuned to a relevant frequency but it is altogether a different matter to actually log the numbers to later cross-check overall drift other than reading it off the screen every 5 minutes and writing it down. Sure, writing it down and then later entering it in a spreadsheet works for almost everyone when you are only doing it once - or at most spasmodically. My needs are a little more concentrated due to the above-mentioned projects and I just don't have the time, inclination or availability to sit watching a screen for hours and hours.

At this point I must categorically state that this project in no way replaces Spectran or Digipan for normal operations BUT it does help you determine how good/bad the actual performance of your equipment might be in regards to frequency drift - and therefore the overall accuracy of any ' effective frequencies' measured with these other applications..

If you have a repetitive need to evaluate frequency drift then a technologically advanced answer is required. I have used the PICAXE series microcontrollers in a few other projects (eg a sequencer) and one of the commands I had seen in the documentation but never used was the "Count" function. I have excerpted the details of this command below.....

From the PICAXE BASIC commands detail : *count* *Syntax:* *COUNT pin, period, wordvariable* *- Pin is a variable/constant (0-7) which specifies the input pin to use.* *- Period is a variable/constant (1-65535ms at 4MHz).* *- Wordvariable receives the result (0-65535).* *Function:* *Count pulses on an input pin.* *Information:* *Count checks the state of the input pin and counts the number of low to high transitions within the time 'period'. A word variable should be used for 'variable'.* *At 4MHz the input pin is checked every 20us, so the highest frequency of pulses that can be counted is 25kHz, presuming a 50% duty cycle (ie equal on-off time).* *Take care with mechanical switches, which may cause multiple 'hits' for each switch push as the metal contacts 'bounce' upon closure.* *Affect of increased clock speed:* *The period value is 0.5ms at 8MHz and 0.25ms at 16MHz.* *At 8MHz the input pin is checked every 10us, so the highest frequency of pulses that can be counted is 50kHz, presuming a 50% duty cycle (ie equal on-off time).* *At 16MHz the input pin is checked every 5us, so the highest frequency of pulses that can be counted is 100kHz, presuming a 50% duty cycle (ie equal on-off time).*	*Example:* *main:* *count 1, 5000, w1 ' count pulses in 5 seconds* *debug w1 ' display value* *goto main ' else loop back to start*

The standard PICAXE-08M runs at 4MHz so please ignore the comments above about 8 and 16MHz in the context of this project.

I set up a PICAXE-08M protoboard with a LM324 quad op-amp (only 2 sections used, and it runs with a 5V supply) as a x30 stage AC-coupled amplifier followed by a straight comparator/squarer circuit (to ensure correct low to high transition pulse shape) and then fed the signal directly into pin 6 of the chip (actual PICAXE function is "pin1"). I fed audio into the first op amp and wrote into the PICAXE some code initially based on the example above and watched the numbers through the debug function on the PC's screen.

The resulting numbers sort-of made sense but I had to manually use a calculator to multiply the high byte by 256, add the low byte then divide by 5 to get the actual frequency. I simply reduced the count "gate" period from 5000 mS (5 seconds) down to 1000mS (1 second) and the values displayed started to get close to what I expected. I added a *binarytoascii* BASIC command to convert the count code into other on-chip registers and I could immediately see

that the count/frequency was just a few Hz low at 1000Hz. I found that I needed to change the gate period number to 1001 or 1002 to get close either low or high respectively. That gave me an updated display of Hz each second as the resulting count. Maximum frequency possible is x9999 Hz.

I really wanted something more accurate than that - preferably with decimals of Hz - well at least one decimal place. I expanded the gate time to 10 seconds (10000mS) and the count now gave me 5 valid numerical digits, the last being tenths of a Hz. By changing the gating value to 10016 (in my case, by fine-tuning with a known input tone frequency), the frequency count was virtually the same as displayed on my frequency counter as sync-ed to my GPSDO standard frequency. I then added a full stop (also known as a decimal point) to the output data stream before the last digit and the PC then showed the frequency accurately - and in human readable format. Maximum frequency possible is 9999.9 Hz. Just remember that the gate period value can be left at a neat 10000 if you don't have an accurate audio tone frequency source because the error will probably be less than 5Hz at 1000Hz anyway. Alternatively you can feed the signal in parallel into both the PC sound card input and the PICAXE 'decoder' and use Spectran to give an on-screen display of the tone frequency and vary the PICAXE's measurement period around 10000 until it reads the same value.

That gave me a serial data stream that I could view in Windows HyperTerminal set at 4800, N, 1. It simply listed the last reading then moved to the next line (newline) before the next one came along. That was a great start - but not particularly useful.

I have written a number of applications in Delphi in recent years (eg my GridLocW software) and this was yet another "local need" that I set about fulfilling. The resulting software reads the serial data (COM port selectable), adds a date and time stamp and displays this info in a small scrollable window. The data is also processed and graphed within the application with the Y-axis scaling automatically changing its frequency limits as the incoming tones frequencies change. Each data sample, whether it be at a 1 second or 10 second rate, is graphed at successive X-axis points with out to about 15000 points available. The data can also be manually exported as a CSV data file and if the Autosave box is checked then it does that automatically every sample period (eg 10 secs) and then every minute into an alternative filename.

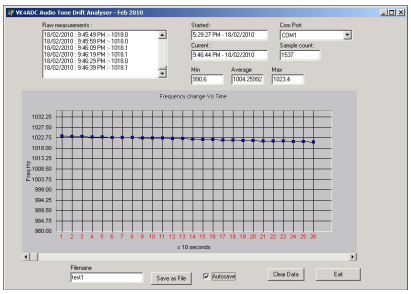
The initial results have actually been quite astounding. A Kenwood TR751A multimode transceiver that I considered was probably reasonably stable (to the ear) simply WASN'T as good as I thought when evaluated. No wonder other people's frequencies are all over the place - and they don't necessarily know because they don't have the facilities- or the knowledge - to evaluate how good - or bad- their radios are in respect of frequency drift. Hey, I'm not necessarily talking about KHz of drift but certainly tens and maybe hundreds of Hz.

I have yet to evalute exactly how drift-free the other radios I have are but now it is a simple process. I simply have to dial up the reference frequency on my Marconi digital signal generator (it being locked to the GPSDO at 10MHz) feed the RF output into the relevant receiver (set to USB) and adjust it's frequency for around a 1KHz beat note, then connect the receiver audio to the input of my audio frequency detector board, connect the serial port and a +12V supply, run my application and it will log the resulting audio frequencies accurately over many hours. If the oscillators don't drift then the line on the graph will be horizontal. If you look at the screen display graphic below then you will see the plot on the graph has a downward slope - i.e. it is drifting down in frequency. On HF, you could use WWV or other frequency standard as the signal source.

One of the things that I have left to do at some future time is to run the likes of 'Spectran' and the PICAXE audio frequency decoder with their audio inputs in parallel so that I can evaluate the long-ish term accuracy of the decoder (ie. that the PICAXE 'count' results do not vary much over time or temperature).

In many cases you need to do the evaluation on a receiver/transceiver from dead-cold so set up the signal source, the power to the interface and have the software running before you apply power to the receiver. Alternatively if you are evaluating an oscillator for drift, have everything else in your test setup frequency-stable before powering up the oscillator. The PICAXE interface and the Windows software will just sit waiting for data.

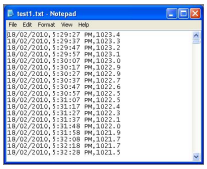
After I saw how things were going, I decided that I would make the audio decoder interface a permanent fixture in my test setup so have done a PCB layout as well as drawn up the schematic so that I have a permanent record and can also make it available to others..



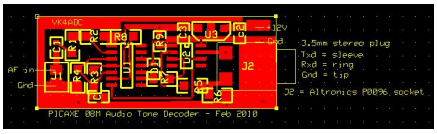
This is my Windows application to process the serial data from the audio decoder board.
The upper LHS shows the raw data while the RHS shows summaries of lowest, average (mean) and highest audio frequencies observed and COM port.
The lower segment shows the graphical results.
The horizontal scroll button allows you to view the subsequent results.

By default the data filename is "test1" and it creates "test1.txt" and "test1 mins.txt" in the executable program's folder as a result.
Of course you can use an alternative name (like TR751A or IC7400..) to differentiate between different radios tested and maybe even a date coding filename (eg TR751A 19Feb10). Normal Windows naming conventions apply.

A larger view can be observed by rolling your mouse over the graphic image above



This is the detail in part fo the "test1.txt" CSV file showing every sampled data.
Last column is sampled frequency in Hz.

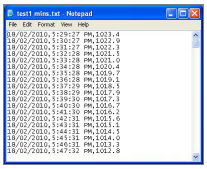


This is the PCB layout designed for top mounting of the components, LM324D (SOIC14) and the PICAXE-08M (SOIC8) devices, a 78L05 voltage regulator plus some 1206 resistors and 0805 capacitors.

The 3.5mm stereo jack on the RH end of the board matches up with the standard PICAXE eval board wiring.

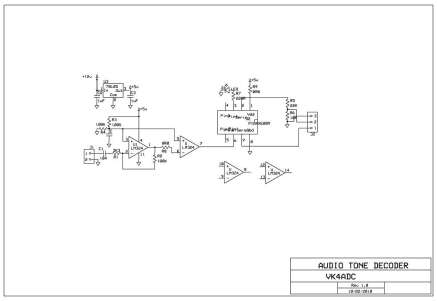
The audio input is on a 2 pin 0.1" header. If you are going to mount the PCB in a jiffy box, the input output and supply connections soldered on the PCB should all be 0.1" headers hardwired to the box-mounted external-world connectors.

A larger view can be observed by rolling your mouse over the graphic image above



This is the detail in part fo the "test1 mins.txt" CSV file showing only every minute's sampled data.
It was added so that there were less points to graph in Excel.

The "time creep" because of a slightly longer gate time than the neat 10000mS plus the serial data processing time becomes more obvious when you see that the time in column 2 occasionally moves to the 11th second instead of just moving by 10 seconds.



The schematic of the audio tone decoder interface is really very simple and uses only a few SMD-style parts.

A larger view can be observed by rolling your mouse over the graphic image above

You don't need to use the SMD-style components to hard-wire this up on veroboard.

Simply use the full-sized DIP14 LM324 IC and the DIP8 PICAXE-08M (use of IC sockets recommended) and standard 1/4 watt resistors and polycarbonate capacitors, maybe paralleled with a 4.7uF or 10uF 16v-25v tantalum at the 3 terminal regulator input & output.

The circuit is simple with only a few wires to be run. Note that there are 2 resistors marked 0R0 on the schematic and these are simply used as links to cross tracks on the PCB layout and are not actually used if hard-wiring.

If you have an interest in making up one of these interfaces for a similar use, drop me an email and ask for the original files.
I have the board layout in ExpressPCB (http://www.expresspcb.com), the schematic in ExpressSCH (http://www.expresspcb.com), the PICAXE -08M BASIC listing and my Windows application (for Win2K & later).

Don't bother me if you just want the code and aren't intending to actually build it up.

The total project was a one-day wonder : building the original proto-board, writing and refining the PICAXE code, writing and refining the Delphi application and doing the schematic and PCB layout all happened in a period of about 8 hours.

Post-construction note :

I fed paralleled audio from a receiver into a PC's sound card input and ran Spectran & the same audio into the audio decoder board and also ran my application software on the same PC to track what it measured, overlapping the screens so I could see both windows similtaneously. The results showed that while the Spectran software provided an instant frequency readout & a waterfall display, the picaxe audio decoder hardware provided effectively the same average value for the 10 second sample period. I even altered the picaxe sample period back to 1 second so that I could track the 2 outcomes more quickly - and the results were extremely close. The main advantage of my picaxe hardware is that you can track the average frequency (/ drift) over far longer periods (eg 24 hours) and use the exported CSV data (the files described above) to create your own graphs in Excel or other spreadsheet software.