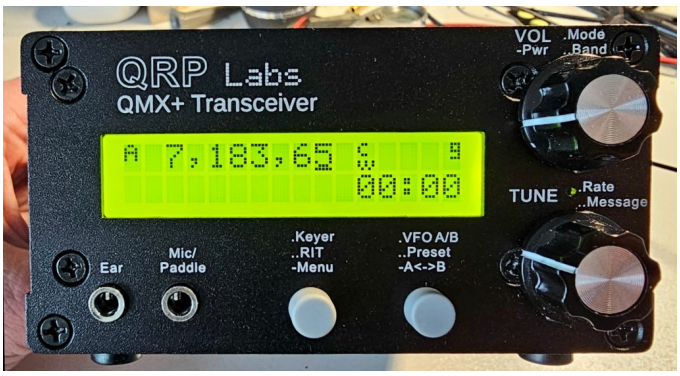
Bringing SSB to QMX

Hans Summers, G0UPL
http://qrp-labs.com
Email: hans.summers@gmail.com

Introduction to QMX and QMX+

QMX is a multi-band, multi-mode QRP transceiver launched at Dayton FDIM in May 2023. It was followed a year later at FDIM in May 2024 by its big brother, the QMX+.

Both transceivers have the same electrical design, same firmware, same performance. But the QMX+ is in a larger enclosure, is an easily built kit, and covers all bands from 160m to 6m. There is an internal GPS module option, a battery-backed Real Time Clock, and plenty of space inside for modifications for which a Dev kit PCB is available. The QMX on the other hand, is very tiny and highly suitable for portable operations including SOTA and POTA activations; it is a very compact build and is available in three different band versions: 80-20m, 60-15m or 20-10m. Both transceivers are available either as kits or assembled/tested transceivers.



The two transceiver brothers have been very well received by QRP operators, and around 9,000 have been sold around the world, to date. They offer a very high and unrivaled ratio of performance and features to price.

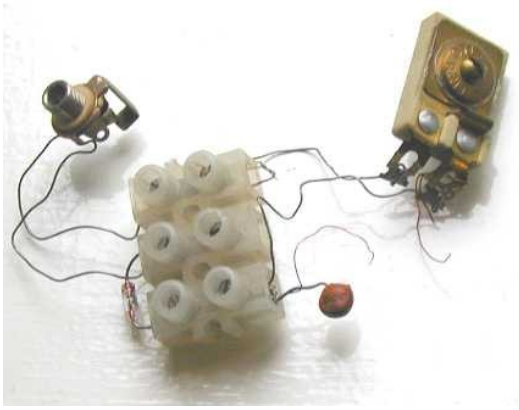
The transceivers use an embedded Software Defined Radio (SDR) design, with very high performance components such as the 110dB SNR Analog to Digital Converter. They have a built-in 48ksps 24-bit USB sound card and Virtual COM Serial port so interfacing with a PC for digital communications is easy using just a single USB cable. The “brain” of the operation is a powerful 32-bit ARM Cortex M4 microcontroller running at 168MHz and including floating point units and Digital Signal Processing (DSP) instructions.

At the initial release, the supported modes were CW and Digital modes (FSK only, suitable for WSJT-X modes like WSPR, FT8, and others including RTTY and JS8). It was always the intention to provide SSB capability later. A built-in microphone is included in the kits and they were also designed to accept a plug-in microphone at the CW paddle port. The amplifier is non-linear and the plan was to use the Envelope Elimination and Restoration technique (a.k.a. Polar Modulation) to achieve SSB transmission. For this purpose an amplitude modulator was included in the design, which was initially just used for CW envelope shaping.

In other words, the design was thought to have sufficient hardware capability and a powerful enough microcontroller, to be able to implement a reasonable SSB transmitter. That was the theory and the dream. The “stretch goal”, which everyone took as a “promise”. Making it into reality, is a whole different story. This article is about that journey, the 9 month development marathon that led to an excellent SSB implementation in the QMX.

In this document I use the terms QMX and QMX+ interchangeably because the SSB project firmware applies equally to both transceivers.

Digital Signal Processing basics



When I was small, I built analog electronics things. I started at about age 6 with a crystal radio receiver birthday gift (see right).

When I discovered digital electronics as a teenager, I was amazed. Everything with logic gates was so, SO much easier! Everything was either a '1' or a '0', nothing in between, and it all followed simple rules of logic. I loved the certainty and had a lot of fun building up really complex logic gate projects.

Later when I started in amateur radio – and all my radio transceivers always have been homebrew only – I was back to analog again. It took me quite a while to put microcontrollers in my projects and even longer to start to contemplate accepting that most wonderful and SCARY of modern concepts: Digital Signal Processing!

The first QRP Labs transceiver design was QCX in 2017. It was an all-analog transceiver but an ATmega328 processor implemented a lot of helpful features such as adjustable frequency and volume sidetone, control of the PLL synthesizer chip (Si5351A), iambic keyer, CW decoder, VFO A/B, messages and frequency presets. A whole suite of built-in test tools was included that enabled initial adjustment and some debugging to be done with no additional equipment.

I began to see that including a microcontroller let me do a lot of stuff that would have required a lot of circuitry to do in analog. Several defining moments are memorable such as when there were faults with the sidetone including a terrible click at the start and end of the sidetone. I was able to resolve this and similar issues solely via firmware updates. I started to understand the power of SDR. You can roll out upgrades to existing radios without any modifications or hardware changes, just a firmware update! Still, the QCX was NOT an SDR, it was more of a Software Assisted Analog Radio.

A year later I embarked on the over-ambitious QSX project, to design an all-mode all-HF 10W transceiver around an embedded SDR architecture. It turned out to be too big a bite to chew on and the pressure surrounding its development was also an issue for me.

In 2021 I launched the QDX Digi modes multi-band miniature QRP transceiver which was the first proper SDR product of QRP Labs.

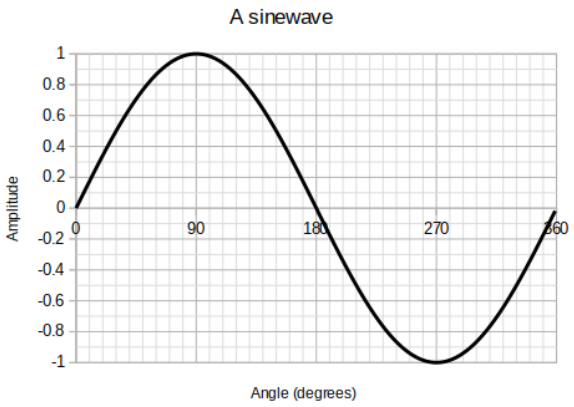
So here are a few key points you need to know about Digital Signal Processing, a dummy's guide or at least an introduction to a dummy's guide, written by the dummy himself.

Sampling

Firstly here's the most fundamental, key difference. The real world is analog. We speak, air pressure waves impact upon a microphone, which converts them into a voltage, which our radio transmitter converts to RF and emits from its antenna. Signals are a continuous changing value (such as voltage). The far distant station answers us, radio waves hit our antenna, and the radio receiver filters and amplifies these tiny voltages, then

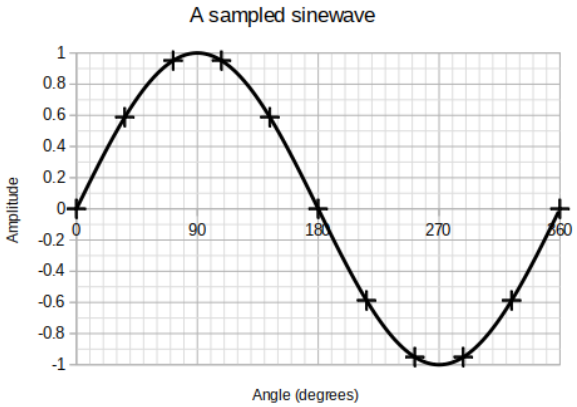
presents them to our loudspeaker; where again, analog voltages move a real world coil, which moves a diaphragm, that creates some air pressure waves which our ears hear as sound.

Digital Signal Processing, on the other hand, operates on discrete samples of the analog world. Numeric measurements of those voltage values at regular instances in time, expressed with a limited resolution. An Analog to Digital Converter (ADC) is used to convert analog voltages into a digital measurement. A Digital to Analog Converter (DAC) is used to convert back the other way, from digital to analog.



Right: here's an unremarkable, but impeccably and beautifully perfect, sinewave.

Now, if we use an ADC to sample it, we will end up with a sequence of numbers representing the sinewave. I have marked those on the sinewave with a little cross, for each one. You can see that I'm showing 10 samples for the complete sinewave. As a stream of numbers, the sinewave is now:



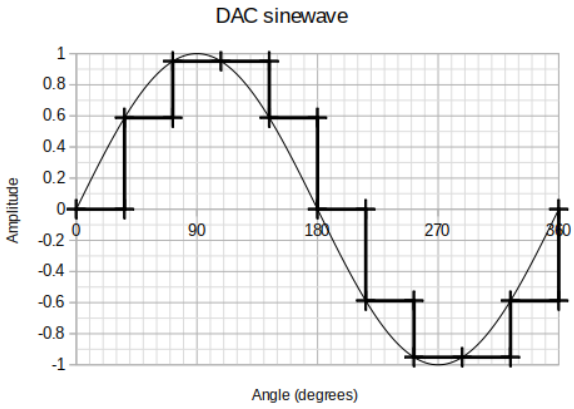
0
0.587785252292473
0.951056516295153
0.951056516295154
0.587785252292473
0
-0.587785252292473
-0.951056516295153
-0.951056516295154
-0.587785252292473
0

Digital Signal Processing can now run those numbers through various calculations to filter, analyze, amplify and otherwise process the signal according to our need.

You'll perhaps notice that I expressed the x-axis of the sinewave graph in degrees. If I tell you that the frequency is say, 1 kHz, that would mean 1,000 cycles per second. Then we could equally well have labeled the x-axis in units of time; the complete cycle would take 1/1000'th of a second, or 1 millisecond.

Sticking with my example, there are 10 samples in the sinewave, and the sinewave is 1000 Hz, that means that there are 10,000 samples per second (sps). We call this the "Sample Rate". Equivalently, we can say 10 ksps (kilo samples per second).

Now what happens if we use a Digital to Analog Converter (DAC) to convert our samples back to analog? See right: THIS is what happens! Oh yuck, that doesn't look much like our original sinewave at all, does it? It is more like a staircase than a smooth sinewave.



But don't worry! Because we can follow this by an (analog) low-pass filter, to smooth out all those square edges, and we'll end up with exactly the sinewave we started off with (assuming we can design a perfect low-pass filter. This is called a **"reconstruction filter"** and it appears after the DAC to get rid of these staircase steps. Conversely ahead of the

ADC we need an **“anti-alias filter”** otherwise higher frequency components will be converted just as if they were part of our wanted signal.

Don’t let anyone tell you that there is any such thing as an all-digital radio! You ALWAYS have to have these analog filters on the way in, and the way out, to smoothly convert between the real analog world and it’s digital representation.

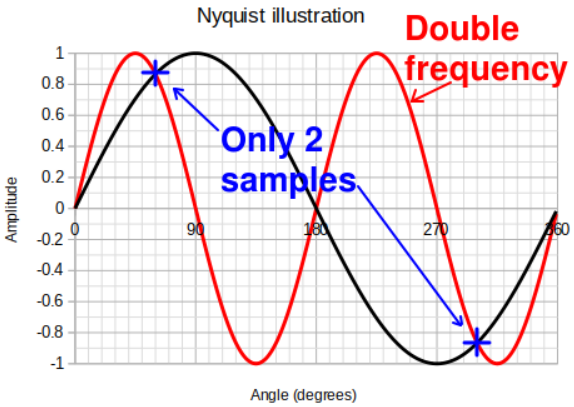
Nyquist

So you might wonder, what if the frequency is higher than 1 kHz, but the sample rate is still 10 ksps. Well, then the number of samples in each sinewave cycle will be less than 10, the higher in frequency we go. But how high can we go?

Harry Nyquist (1889-1976) was a Swedish dude, a physicist and electronics engineer, who is very famous in communications theory, for the “Nyquist frequency” concept named after him. Namely, that any frequency which is LESS than the Nyquist frequency, can be perfectly reconstructed into analog without any distortion or unpleasant artifacts.

The Nyquist frequency is HALF the sample rate! In this case, 5 kHz.

We can quite easily visualize WHY this is the case. If we have a sinewave which is of such high frequency (say 5 kHz) and that at the sample rate (10 ksps) there are only 2 sample points in each sinewave, then we have lost information about the signal. A sinewave (and in fact any number, an infinite number) of sinewaves at harmonic frequencies could also be drawn on the graph, and they would pass through the same 2 points. We have no way of knowing which is the correct one.



If the frequency being sampled is LESS than the Nyquist frequency (5 kHz), then each cycle will contain MORE than 2 points, and only ONE sinewave, the correct one (whose frequency is less than 5 kHz) can be drawn that will connect all the sample points.

The Nyquist frequency is one of the most important concepts in DSP! This is also why we need the anti-alias filter ahead of the ADC. Because otherwise all the harmonics of the wanted frequency, will also be converted by the sampler, and look like frequencies that are below the Nyquist frequency (a.k.a. “aliases”).

Resolution, Dynamic Range and Signal to Noise Ratio

Another extremely important concept is that of the resolution of the numeric representation of the analog signal. It’s the precision of the step height of the formerly diagrammed “staircase”. If the number is not accurate enough, then when it is reconstructed into an analog signal, there will be distortion and spurious signals.

But everything is an engineering trade-off. Precision costs money. High resolution ADC and DAC chips can get very expensive. Additionally the more precisely you process something, the more computational power it demands; then you need a more powerful (fast) microcontroller: which also costs more money, and consumes more current. So you have to decide how to set the design goals.

Dynamic range is the difference between the largest signal which can be handled by the system without distortion, and the smallest signal which can be handled. The “smallest” signal is normally determined not by the smallest number which can be represented, but by the noise floor of the system. So dynamic range of an ADC is really almost the same thing as Signal to Noise Ratio (SNR).

The rule in DSP is that each 1-binary bit of resolution, is equivalent to 6 dB of dynamic range. Which also makes sense because the formula for dB is 20 * log (voltage ratio). Since binary is a base-2 system, you can imagine that each extra bit halves the voltage per bit, doubling the resolution. And 20 * log (2) = 6.02 dB. Which is where the 6 dB rule comes from. So everything works out.

In a radio receiver, a 100 dB dynamic range is considered a good target to achieve, as a rule of thumb, by reasonably modern radio receivers. 16 bits would be 16 x 6 dB = 96 dB dynamic range. So a 16-bit system could, theoretically, almost achieve a reasonable level of performance. Indeed some SDRs run all their DSP calculations in 16-bit integer arithmetic which makes the calculations much faster than if they had been done in the higher precision floating point format.

In reality it's quite hard to position that "window" of dynamic range such that it perfectly fits the range of signal levels you expect to see at every stage in the signal chain. In QMX/QMX+ a powerful microcontroller is used, having a built-in floating point processor in hardware; therefore everything is processed in floating point format which is extremely precise, many many orders of magnitude more precise than we could hope for from the ADC or other parts of the hardware. So the problem of calculation accuracy is eliminated, it is no longer a factor limiting the performance of the radio receiver.

ADC chips have a noise floor too. Don't be amazed just because someone tells you an ADC chip is 24-bit! Or even that it is 32-bit! In reality the effective number of bits is much smaller; the real dynamic range is the difference between the largest signal that can be represented without overflowing the ADC output number, and the noise floor of the ADC chip. For example, the ADC chip used in the QMX is PCM1804, a 24-bit chip. 24 x 6 = 144 bits… but the dynamic range specification of 110 dB. 110 dB equates to a little over 18 really useful bits. The other 6 lesser significant bits are just noise.

Summary:

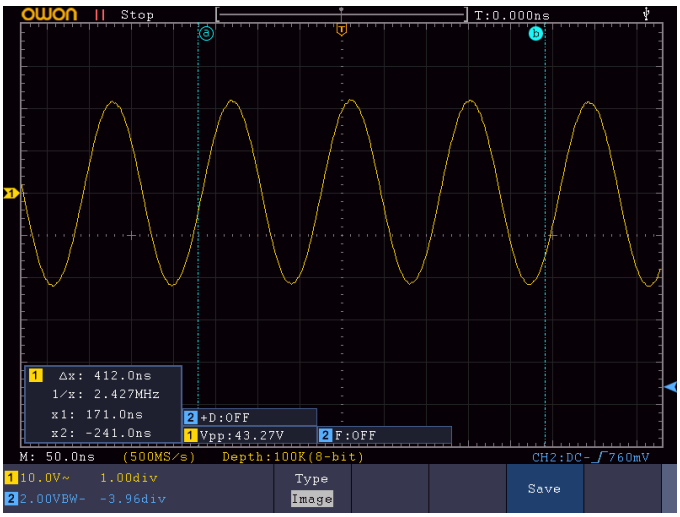
- Sample rate: How many numeric samples we convert to digital, and process digitally, per second
- Nyquist Frequency: for any given sample rate, the Nyquist frequency is half of it; and we must not give the system any frequency higher than the Nyquist frequency!
- Bit resolution: the number of bits of resolution of the analog to digital conversion; more precise is better, and we really need 17 or more bits to break the magic "100 dB dynamic range" rule of thumb for a good radio
- Dynamic range: The difference in dB of the largest signal that can be handled without gross distortion, and the smallest one; since the smallest one is usually determined not by the smallest binary bit, but by the noise floor, dynamic range is also usually the same thing as the signal to noise ratio of the conversion.

Software Defined Radio basics

There are two types of SDR receiver. The Direct Digital Conversion (DDC) type such as the Elecraft K4, or Icom 7300, digitize a wide chunk of spectrum, 10 or more MHz, and process it all digitally. This requires advanced ADC chips and a relatively powerful amount of processing, to do all that calculation.

The other (older, perhaps) type is baseband SDR. The RF signal is converted to "baseband" (which means, direct conversion to 0 Hz) by two mixers with a 90-degree offset local oscillator. Or equivalently, a Quadrature Sampling Detector (QSD, a.k.a. Tayloe detector) which is essentially the same thing as two mixers. These produce so-called "I and Q" signals. Now if we apply a 90-degree relative phase shift to these signals, the unwanted sideband is shifted by 90-degrees in one direction and the wanted sideband is shifted by 90-degrees in the other direction. Adding these two signal paths will result in a doubling of one sideband, and cancellation of the other. Hence we can eliminate an unwanted sideband and demodulate the wanted sideband, in other words: achieve SSB reception!

The QMX transceiver is of this type (and others such as Elecraft KX2, KX3 and the mcHF). The original QCX transceiver I mentioned previously, has an all-analog receiver but it also does exactly the same thing, just in hardware rather than DSP. There's a QSD for conversion to baseband I Q, and an op-amp phase shift network for the 90-degree phase shift.

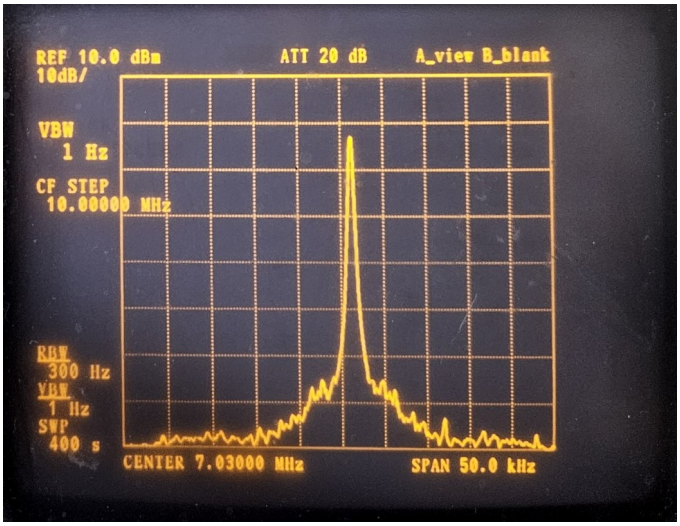


Relatively speaking, a DDC type transceiver requires expensive ADC and powerful processor, which also translates to more money, more space, and more (much more) current consumption. Most of these undesirable characteristics are the enemy of the QRP ham who wants small, inexpensive, lightweight, battery-friendly equipment. DDC transceivers do not even present a performance advantage; the main benefit appears to be the wide spectral view, having a nice large colour panadapter style display is an attractive bells-and-whistles feature.

Time domain, frequency domain

An oscilloscope, a very useful item of test equipment with which many of us will be familiar, and perhaps own, is an example of viewing a signal in the TIME domain. This just means that the x-axis is time. The vertical axis is amplitude. So a sinewave such as in the above discussions, just looks like, well, a sinewave. So that's easy.

A spectrum analyzer is a more exotic and typically more expensive piece of equipment, for viewing signals in the FREQUENCY domain. This again simply means that frequency is now the x-axis, and the y-axis is amplitude. It's like a swept frequency receiver, which records the signal strength at each frequency in its sweep. Spectrum analyzers are useful for many purposes, the most common is checking the harmonic content of transmitters. Another important use, particularly in the context of this article, is for SSB transmitter two-tone testing!



A single tone sinewave just looks like a single spike on the spectrum analyzer (see right).

Note that the y-axis of a spectrum analyzer is normally shown in decibels (dB), because these instruments are very precise and sensitive with a large dynamic range.

The cost of these instruments has definitely become a lot more affordable in recent years. There's even a small one called "TinySA" available on AliExpress, Amazon etc which costs only about $50 yet has quite reasonable performance. It's plenty adequate for things like transmitter harmonic testing and I highly recommend it, though it does not have sufficient resolution for SSB transmitter two-tone testing.

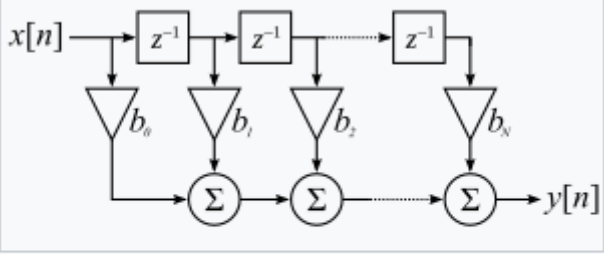
Fourier transform

The Fourier transform is a mathematical calculation which converts a time domain signal into a set of frequency buckets; effectively it does a similar thing to the swept frequency spectrum analyzer, but mathematically. This is what you see on the panadapter of your expensive radio. At each point the radio has analyzed a wide chunk of signal and split it into small buckets, then shows you the amplitude in each bucket.

There's a faster version of the Fourier transform called the Goertzel algorithm, which is a reduced calculation for determining the amplitude of a single frequency bucket. This is used in the QCX and QMX transceivers for applying a bandpass filter for the CW decoder.

Finite Impulse Response filters (FIR)

Two types of digital filter are common and are used in QMX and other transceivers. The first (and easiest) type is the FIR (Finite Impulse Response) filter. This FIR filter sounds very complicated but really it isn't.



A direct form discrete-time FIR filter of order *N*. The top part is an *N*-stage delay line with *N* + 1 taps. Each unit delay is a z^{-1} operator in Z-transform notation.

Conceptually there is a delay line of signal N samples. And there's a corresponding vector of N+1 coefficients. At each instant of time, we put one new sample into the delay line. Then we calculate the output sample, as the total sum of each delay line sample multiplied by its coefficient. This can implement various filter types.

Above right: here's a diagram I stole from Wikipedia at https://en.wikipedia.org/wiki/Finite_impulse_response

$$y[n] = b_0 x[n] + b_1 x[n-1] + \cdots + b_N x[n-N]$$
$$= \sum_{i=0}^{N} b_i \cdot x[n-i],$$

That formula is nothing to be scared of, it's a reasonably simple matter. There is just a delay line of buckets (containing the sample numbers), our N samples. And a set of corresponding coefficients we multiply them by. At every instant in time, just multiply each sample with each coefficient, and add everything up.

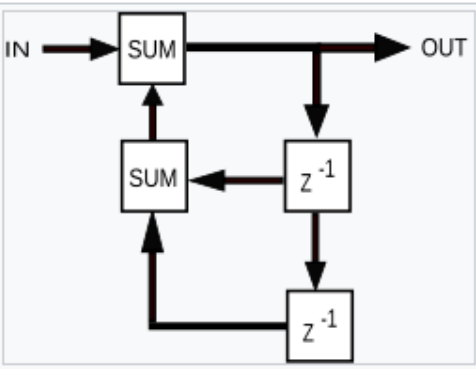
The number N+1 is known as the number of "taps" of the filter.

The advantages of this kind of filter are that there is no feedback so there is no possible instability, the output range will always be bounded and easily predictable. It's also possible to design filter coefficients that have useful properties such as linear phase response – which as we will see later, will be important (for the Hilbert Transform). A disadvantage is that a relatively large number of taps are typically needed and this means the filter is quite slow to calculate.

The FIR filters are quite easy to implement; the difficult part is designing the set of filter coefficients. However fortunately there are tools available for filter design, which spit out a list of coefficients and plot the response for you. Or, there are numerous examples of filter designs in people's projects all over the internet.

Intinite Impulse Response filters (IIR)

This type of filter is a little more complex because it involves feedback. This image is also from Wikipedia: https://en.wikipedia.org/wiki/Infinite_impulse_response



An example of a block diagram of an IIR filter. The z^{-1} block is a unit delay.

With a like-for-like filter slope and cut-off characteristics, an IIR filter has far fewer coefficients than the equivalent FIR filter, making the calculation much faster.

In QMX, IIR filters are used wherever possible. For things like the audio filtering which determines the selectivity, and where there is no need for linear phase response, IIR filters are very useful. Again, there are many sources of coefficients or design tools to generate them.

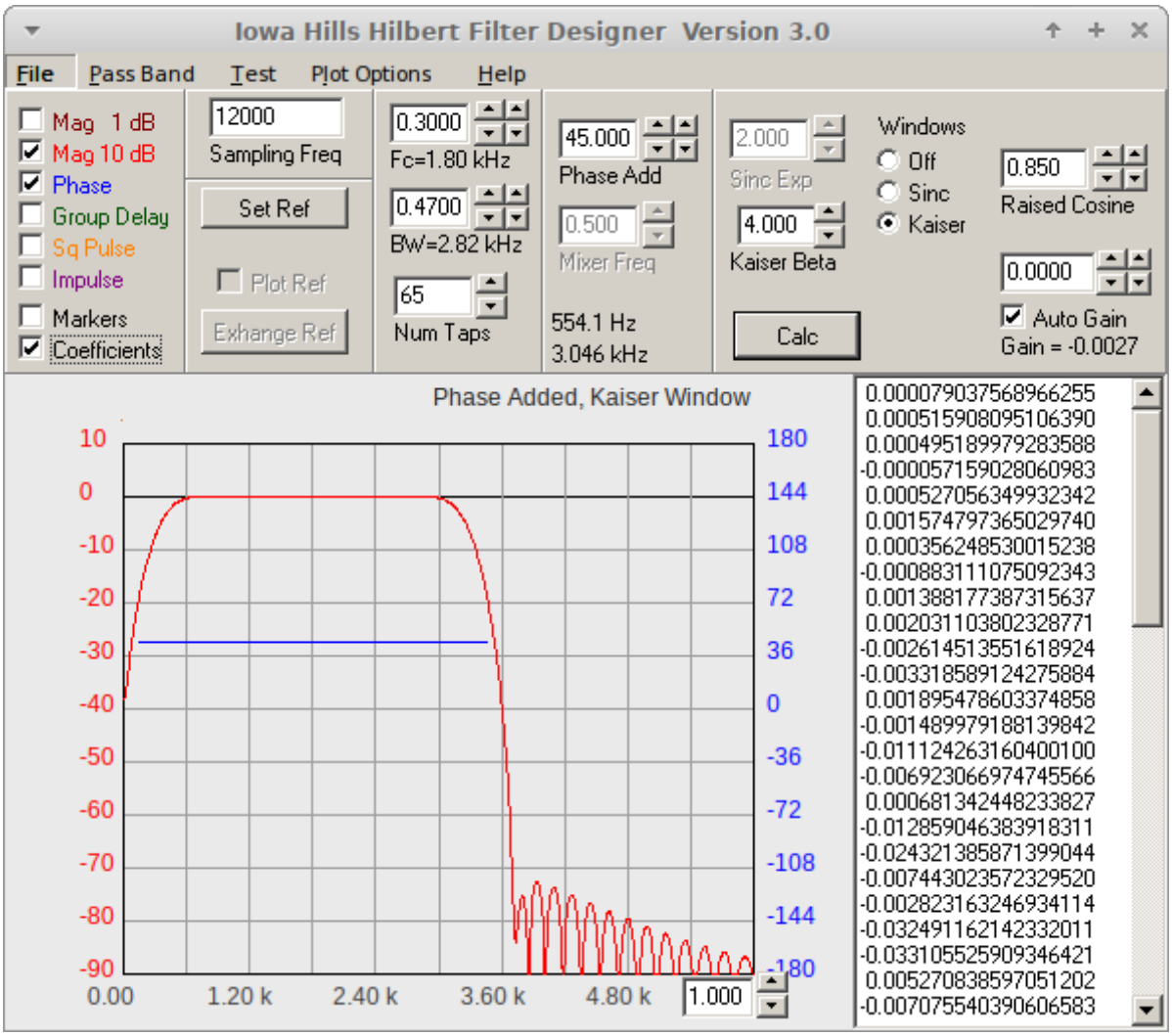
Whole books are written on the design of IIR and FIR filters. The above is only a very very brief introduction.

Hilbert Transform

A German mathematician, David Hilbert (1862-1943) is another dude whose name comes up a lot in this area, because this whole Hilbert Transform that is so crucially important for SSB, is named after him!

The Hilbert Transform is a special FIR filter, whose coefficients are designed to produce a 90-degree (or other) phase shift to all the frequency components across a wide audio band to cover the bandwidth of an SSB signal.

There's a particularly nice set of filter design programs designed by Iowa Hills Software LLC, licensed as free software. For some reason the web page which hosted the software disappeared, but the Internet Archive a.k.a. "Wayback machine" has copies of all the pages, including the downloadable executable files! It's also a Windows application, whereas I use Linux; but it runs fine under WINE (a program for making Windows programs run on Linux).



The above screenshot shows what the Iowa Hills Hilbert Filter Designer looks like. You can experiment with the amplitude response, and set the desired phase shift and all other parameters. It allows you to play with all the parameters and calculates the set of coefficients. It's then a matter of copy and paste to put these coefficients into the source code.

Some SDRs use a 90-degree Hilbert Transform in either the I or Q path, and then they generate a corresponding zero-degree Hilbert Transform for the other path. The point of the latter, is to apply the same amplitude response, and to ensure that the delay as the samples pass through the delay line, is equal in each path. Personally, I prefer the idea of things being symmetrical, it never seems to hurt and sometimes even helps. So my two filters are for -45 degrees and +45 degrees; all that matters is the RELATIVE phase shift applied to I and Q.

The surprisingly simple anatomy of an SDR receiver

Given the above building blocks, one is then at the stage where a complete SDR receiver can be designed.

1. Hardware: We first need an analog band pass filter to reject strong out of band signals
2. Hardware: A Quadrature Sampling Detector to mix the RF signals down to baseband I and Q signals
3. Hardware: A low pass “anti-alias filter” because we don’t want signals above the Nyquist frequency aliasing into our desired passband
4. Hardware: An ADC chip to convert the analog I and Q signals into digital representations. Remember ideally we want at least 100 dB of dynamic range which means at least 17 effective bits of resolution.
5. Software: Hilbert Transforms (an FIR filter) acting on the I and Q channels to impart a 90-degree relative phase shifting
6. Software: Summation to add the resulting two paths for Upper Sideband (USB) or subtract for Lower Sideband (LSB)
7. Software: A nice sharp audio filter (IIR filter) to obtain the desired audio passband
8. Hardware: A DAC chip to convert the digits back to analog voltages
9. Hardware: A reconstruction filter to smooth out the steps of the staircase
10. Hardware: Some kind of audio amplification to drive headphones or a loudspeaker

Two things are very striking about this simple SDR receiver.

Firstly, look how much of it has to be done in hardware! This is a common misconception: that an SDR software does everything digitally! It does not, there are still many parts of the signal chain that are necessarily analog circuits.

Secondly, the actual Digital Signal Processing involved in SSB reception is relatively simple, and this was very surprising to me. It’s not as scary as I thought. I spent a lot of time developing the firmware for QMX, but the SSB reception was by far the easiest part. Plenty of other things like the switching power supplies, getting clean CW FSK, proper keyer timing, and a bug-free user interface, all these things were very much more difficult than the DSP parts of the project. For SSB **reception**, this is...

QMX has supported Digital modes from the start, and the receiver IS an SSB receiver, in Digi modes. The transmission part of the Digital mode in QMX is by measuring the audio frequency from the PC and generating a single carrier tone at the correct frequency – which I have described extensively previously.

I came to really love the concept of Software Defined Radio and decided QCX would be the last analog radio I would design – at least, as a product. I’ll certainly make my own crazy analog homebrew projects but for a product, SDR has so many advantages. The performance to cost ratio is much higher than an analog radio, and the most amazing aspect, being able to easily distribute firmware updates to provide improved functionality and bug fixes, is a really major benefit. How else could I have launched QMX in May 2023, with the rumor of SSB attached to it, and after the lapse of 21 months, he able to finally provide it to 9,000 QMX owners via an easy-to-install firmware update?

Polar Modulation SSB background and heritage

Conventional SSB exciters have an SSB modulator (whether done in analog circuits or using SDR techniques) followed by a linear driver amplifier and a linear power amplifier. The linearity is critical to the performance; non-linearities cause intermodulation products which produce close-in spurious products outside the desired SSB passband, causing interference to other spectrum users (known as "Splatter"). The design of the amplifiers needs a lot of care and attention, and the problem is multiplied when a multi-band unit is targeted. It is difficult to achieve good linearity.

In 1952 Leonard Kahn proposed in an IEEE paper, an SSB generation method called EER (Envelope Elimination and Restoration). Kahn's paper is available here: https://qrp-labs.com/images/qmx/ssb/Kahn1952.pdf

In his paper Kahn describes splitting the SSB process into phase and amplitude components. Phase modulation and amplitude modulation are applied separately. The phase modulated signal is amplified to full power by a non-linear RF amplifier (such as Class-C); then an amplitude modulator is applied to the supply of the RF amplifier to effect the amplitude modulation.

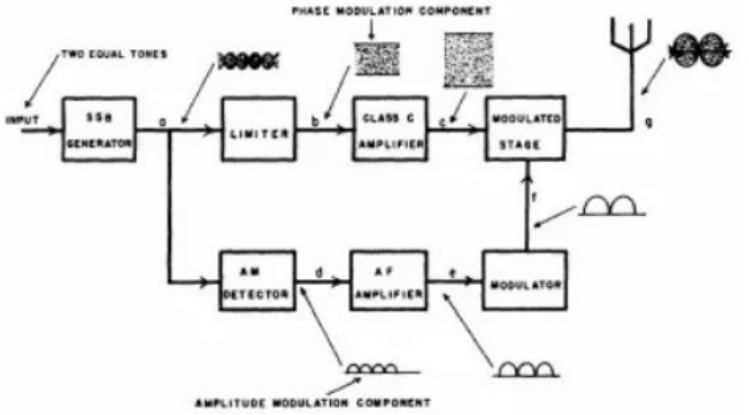


Fig. 2—Simplified block diagram of system.

The main benefit of this approach is higher performance because RF amplifier linearity ceases to be an issue. This is explicitly stated by Kahn in the paper.

Much fanfare has been made subsequently of the higher efficiency of such non-linear amplifiers but it should be noted that obtaining the full benefit of higher efficiency would require high efficiency amplitude modulation also (perhaps using PWM switching techniques).

This paper https://oa.upm.es/7900/2/INVE_MEM_2010_79469.pdf further explores high efficiency techniques for modulating amplitude using multi-level converters for the amplitude envelope amplifier. But this paper is not really so relevant to us here, since it does not deal so much with EER itself, it is focused on how to set up a series of efficient stepped regulators for the amplitude modulation.

EER implementations start off with an existing SSB signal, then strip off its amplitude, detect the phase, and apply phase and amplitude modulation as two separate paths. It's debatable whether this is exactly comparable to what is going on in QMX. In QMX audio baseband signals are converted to polar coordinates (phase angle and magnitude), which are then modulated; the phase modulation is applied by a series of rapid frequency updates to the MS5351M frequency synthesizer chip, and the amplitude modulation is applied in the conventional way by modulating the supply voltage to the non-linear amplifier. "Polar Modulation" is a term that I think I prefer to EER, in the QMX context. Though you could argue that the microphone audio signal is equivalent to an SSB signal at baseband (zero carrier frequency) so you are still eliminating and restoring its envelope.

In 2017 Brian K1LI and Tony K1KP published a QEX article "The Polar Explorer" see https://www.arrl.org/files/file/QEX_Next_Issue/Mar-Apr2017/MBF.pdf describing the design of a high power SSB transmitter using Polar Modulation (EER) techniques. A 500W version of the transmitter

was exhibited at Dayton hamvention in 2019 and I had the opportunity to see it for myself. This was my first introduction to Polar Modulation (EER). See also the Polar Explorer website https://www.polex-tech.com/.

The article is very interesting; of particular note the phase pre-distortion and phase pre-distortion techniques employed in the transmitter. The article describes construction of a lookup table for the phase error using an AD8302 gain and phase detector chip - At Digikey even in 2500pcs reel quantity the price is still $18.68 + taxes, OUCH! I am particularly proud of developing a way in QMX to measure the phase error without any additional hardware. More on this in a moment.

In 2019 Guido PE1NNZ presented his special firmware and extensive hardware modifications to the QRP Labs QCX transceiver http://qrp-labs.com/qcx to turn it into a multiband SSB transceiver using Polar Modulation (EER). This was a derivative work on his earlier (2013) project to generate SSB directly using a Raspberry Pi, see https://www.pe1nnz.nl.eu.org/2013/05/direct-ssb-generation-on-pll.html. The QCX-derived SSB transceiver spun off into its own separate project, separate forum. The original page is here: https://github.com/threeme3/usdx. Subsequently there was a lot of development work done by Guido PE1NNZ, Barb WB2CBA, Manuel DL2MAN and others. Several Chinese produced versions of uSDX are very popular and still in production, as well as an "official" version named (tr)uSDX by Guido PE1NNZ and Manuel DL2MAN see https://dl2man.de.

The uSDX/(tr)uSDX transceivers suffer significant performance limitations due to the very minimalist hardware used, including the 8-bit 20MHz ATmega328 AVR microcontroller. Pascal VK2IHL has a very long and interesting page https://vk2ihl.wordpress.com with many details and measurements. QMX achieves a much higher performance due to the investment in much more powerful hardware (32-bit 168 MHz ARM Cortex M4 MPU with floating point and DSP units). This is not a criticism of uSDX, these are simply engineering trade-offs.

Despite the performance limitations, we owe Guido PE1NNZ a huge debt of gratitude for popularizing and further developing the Polar Modulation concept, demonstrating its practical implementation in minimalist hardware. It's a genius work that is the inspiration behind the QMX SSB firmware implementation. Guido is the giant whose shoulders I stand on.

Dave M0JTS has recently written extensively about the Polar Modulation process, including theoretical explorations, simulations, and experimental determination of the frequency settling time of the Si5351A. His work is a very interesting read, see https://daveshacks.blogspot.com/2025/02

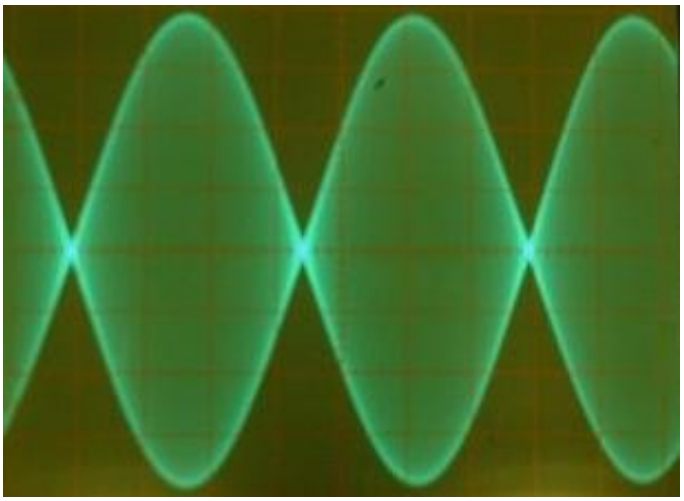
Two-tone testing

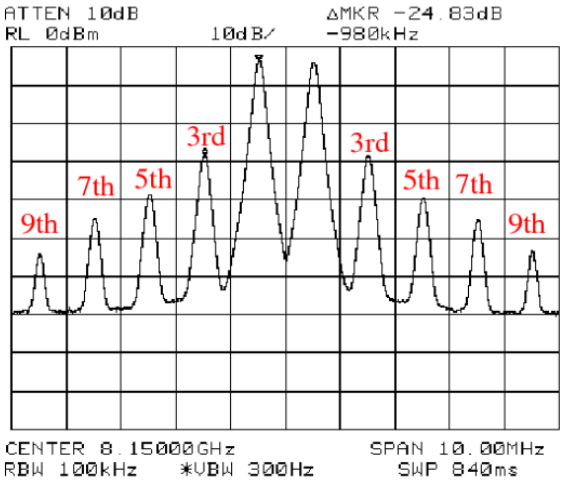
Before getting into the nitty gritty of how SSB is implemented in the QMX, I want to explain the most important test for SSB transmitters. Throughout the development of the SSB firmware for QMX, I was continuously trying new things and running this test.

For an SSB transmitter, the most important measure of its performance is that the signal remains confined to the desired SSB bandwidth and does not significantly overflow into adjacent frequency spectrum which could cause interference to other spectrum users. Any non-linearity in the power amplifier or preceding stages of a conventional SSB transmitter, causes intermodulation products which appear outside the desired bandwidth. These are known as “splatter” among amateur radio SSB operators; it can easily be caused if an amplifier is driven too hard, causing the RF peaks to be clipped which is a form of distortion causing high levels of intermodulation products.

The standard way to test the intermodulation performance of SSB transmitters is called the two-tone test. Two non-harmonically related audio tones within the SSB passband are injected into the transmitter. 700 Hz and 1900 Hz are the near-universally accepted standard tone frequencies for this test. The resulting RF output from the power amplifier to a dummy load, is observed carefully on a spectrum analyzer using a fine resolution setting, to resolve and measure the undesired intermodulation products.

The images below illustrate how the RF output of the transmitter, running a two-tone test, looks on an oscilloscope (left) compared to on a Spectrum Analyzer (right). These are just for illustration, not from a QMX (see later). It also aptly illustrates why it is critical sometimes to use the correct instrument for the job. The intermodulation products are practically impossible to see on an oscilloscope but can easily be measured extremely precisely on a spectrum analyzer.





The spectrum analyzer image (right) identifies the 3rd, 5th, 7th etc intermodulation products. In a two-tone test, the central two peaks are the actual wanted tones. They would be separated, in our case, by 1200 Hz (1900 Hz – 700 Hz). In a perfect world this is ALL we would see on the screen. In reality we see spurious products spaced every 1200 Hz along the screen, these are the odd-order intermodulation products.

Note that there are two different units in common use for reporting IMD levels, and it can get unnecessarily confusing. One way of reporting IMD products is to report the level of the spurious responses relative to the desired carrier (each of the two tones). In that case the IMD level may be -35 dBc for example (dBc means dB relative to carrier level). The OTHER way is favored by the ARRL in lab measurements in QST reviews (and elsewhere), and expressed intermodulation products relative to PEP (Peak Envelope Power). There is a 6dB difference between the two ways of expressing the same result. So -35 dBc is equal to -41 dB PEP. It

doesn't matter which one is used, as long as you are clear on what you mean, and make sure any comparisons you make are all using the same units. I got used to dB PEP so I use that.

To give some idea of what levels are considered good, in HF amateur radio equipment; opinions vary but a good general guide that I have seen mentioned in several places, for the first IMD product (IMD3):

-10 dBc (-16 dB PEP): Very poor
-20 dBc (-26 dB PEP): Poor
-30 dBc (-36 dB PEP): Good
-40 dBc (-46 dB PEP): Excellent

For some practical examples, I turned to several ARRL QST equipment reviews I have on file. The Elecraft KX2 ARRL lab measurements reported typical IMD3 -36 dBc (worst -30 dBc on 40m). Confusingly, though the ARRL likes dB PEP measurements, these numbers are in dBc in the review. I would like to quote the Elecraft K3S review but it just says dB, it does not even state whether that means dBc or dB PEP! See what I mean, about things getting confusing!

Generating SSB in QMX

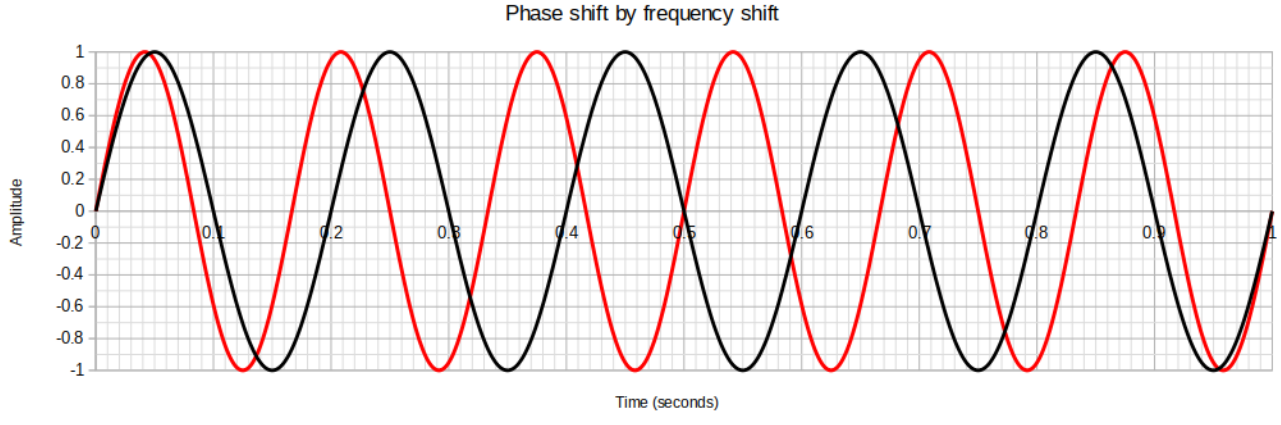
Now finally on to the marathon task of getting SSB to work on the QMX!

The first task was to generate a two-tone test signal. The easiest way to do this seemed to be to use the already existing DDS (Direct Digital Synthesis) module in the code, which was already used for generating a nice sinewave for the CW sidetone. That uses a 16-bit lookup table; later, as I progressed with making the two-tone performance better and better, I realized that the spurious content of the two-tone generation became the limiting factor of the performance and I needed a better two-tone generation system. I created a 120-point set of precise floating point numbers in a spreadsheet and stored them in my code. Now the 700 + 1900 Hz tones were perfect to many decimal places and could never limit the performance.

The whole principle of generating SSB by Polar Modulation is to separate the audio signal into phase and amplitude components, and modulate them separately to RF at power. The phase modulation is done by rapid frequency updates to the MS5351M frequency synthesizer. The amplitude modulation is done by the QMX' amplitude modulation circuit which is designed to multiply the output of the microcontroller's DAC peripheral by a factor of a little over 6, and apply this as the supply voltage for the Class-D switching power amplifier. The RF power amplifier is totally non-linear; the amplitude modulator however, is very linear. It's much easier to achieve good linearity at low audio frequencies, where multiple transistors can be arranged in a feedback amplifier.

Phase modulation via frequency updates

Phase modulation by a sequence of frequency updates, is the key technique demonstrated so cleverly by Guido PE1NNZ in his uSDX implementation. We need to understand how to generate phase modulation by using frequency updates to the frequency synthesizer chip in the QMX, the MS5351M.



The best way I can think of to explain it, is as follows. Imagine two sinewave signals. One has a frequency of 5 Hz, and the other has a frequency of 6 Hz. One second of these sinewave signals is shown in the above simulation.

If we start these signals “in-phase” exactly at time zero, on the left of the plot; then exactly one second later, they will be “in-phase” again, meaning that they will both be at the zero-angle point of their waveform at exactly the same time. But another way of looking at it, is that one signal has gone through 360 degrees of phase shift, relative to the other; the 6 Hz signal’s phase has gone 360 degrees faster, over the course of 1 second, such that it has caught up with the 5 Hz signal.

Like two Formula One race cars around a racetrack. One car has gone faster than the other, and eventually the faster car overtakes (laps) the slower car.

Gradually, over the course of 1 second, the phase shift has shifted linearly from 0 to 360 degrees. Now if I ask you, what about at time 0.5 seconds? Clearly the phase shift at this point, is 180 degrees. And at 0.25 seconds? The phase shift is 90 degrees. In fact, for any given interval of time, we can calculate how much phase shift has occurred.

Now if we used a 2 Hz frequency separation instead of 1 Hz, everything would happen twice as fast. And if we used a 12 kHz frequency difference? Then everything would happen 12,000 times as fast. Perfect for the 12 ksps sample rate I ended up using in QMX for the SSB processing. So for a desired 360 degree phase shift, if we shifted the frequency of the MS5351M output for 1/12,000’th of a second by 12 kHz, we would get our desired 360 degree phase shift.

And in fact, for ANY desired phase shift between 0 and 360 degrees, in a sample interval of 1/12,000’th of a second, we can very easily calculated the necessary frequency shift to apply: it is

12 kHz * phase_angle / 360.

To put it perhaps more formally, the phase shift is achieved by the integration of the frequency shift over the sample time.

The questions naturally arise, how fast can frequency updates be applied to the MS5351M. The question is actually a two-part one. Firstly, how fast can we command new frequencies to the MS5351M, and secondly, can it obey our frequency instructions fast enough.

Via some care in choice of the MS5351M configuration parameters, the set of parameters to update can be reduced to just four one-byte registers. The MS5351M supports “burst-mode” register updates on its I2C interface, and happily the four registers to be updated are sequential, which means that we only need to send the register number of the first one, followed by the data for the next four. Experimentally it can be determined that an I2C bus rate of 1 MHz is capably supported by the MS5351M. Although the datasheet only makes mention of the “standard” 100 kHz and 400 kHz I2C bus rates, that doesn’t equate to a specification of the maximum it will tolerate; the chip is at least 10 years old now and faster I2C “standards” are now commonplace. Experimentally it can be determined that the maximum update rate of the MS5351M, when using fast DMA (Direct Memory Access) techniques, is a little over 16,000 frequencies per second.l at 1MHz bus speed. Now in recognition of the fact that nothing is ever as easy or as regular as it seems, we should have some safety margin and not make anything too over-difficult; so I chose a processing rate of 12 ksps to be comfortably less than the fastest the MS5351M could deal with.

The second question of whether or not the frequency commands are implemented fast enough, has been answered in two ways. Two correspondents independently and voluntarily did some sophisticated measurements of the settling time of the MS5351M and both of them returned a consistent answer, that the frequency updates took approximately 1us To settle at the new frequency. This is 1/83’rd of the sample time, relatively quick, so it seems likely that it should not be a significantly traumatic factor.

The other way to answer the question is simpler, which is to just state carelessly that “the proof of the pudding is in the eating”. So: if the intermodulation products do indeed come out rather low, then we will know that the whole thing works rather nicely and the MS5351M was quick enough doing its work.

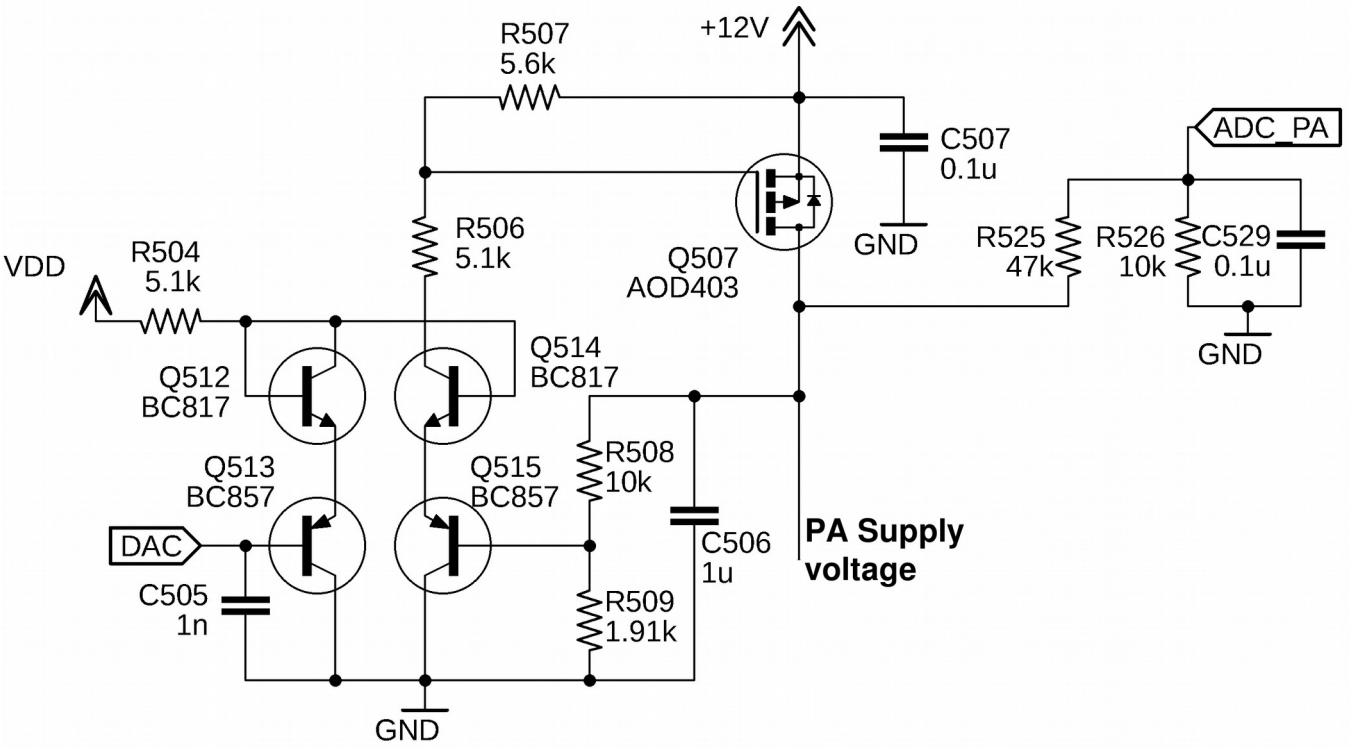
The importance of a fast sample rate

After everything I wrote above about the Nyquist frequency, it would be tempting to assume that it will suffice to run our Digital Signal Processing at a rate such that the highest frequency to be handled will be less than half the sample rate. This was the argument put forward for the uSDX where the sample rate of 4.8 ksps was thought to be just about sufficient, for an SSB bandwidth of up to 2.4 kHz.

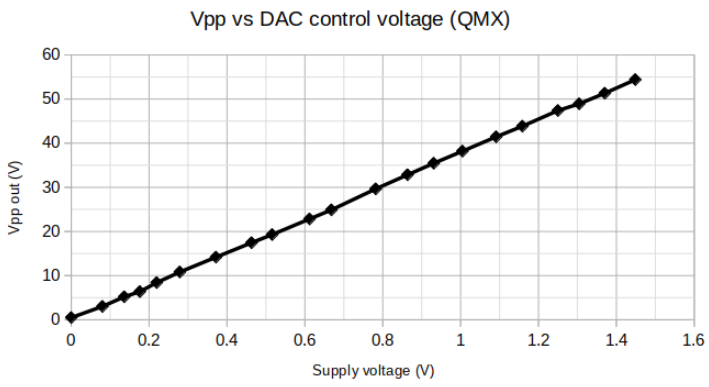
However, this tempting conclusion is not correct. In fact, there is a rule of thumb that the bandwidth of the phase and amplitude modulation, to achieve IMD products not more than -30 dBc, should be at least FIVE times the audio bandwidth. This is further explored in the articles of Dave M0JTS I referred to earlier, see https://daveshacks.blogspot.com/2025/02. By “bandwidth of the phase and amplitude modulation” we of course mean the Nyquist frequency (not sample rate). So for uSDX, 2.4kHz bandwidth is certainly not enough to achieve a reasonable IMD performance. But as I said above, uSDX has many compromises and we know this; it is not the point about uSDX; uSDX showed the way forward and was a tremendous leap of genius by Guido PE1NNZ.

QMX uses 12 ksps for the DSP rate as we are limited by the 16 kHz update rate of the MS5351, and 12 is as close as I dared to go, just now. A future experiment might be to try to increase it further.

Amplitude modulation



This is the amplitude modulator circuit in QMX. There’s a 12-bit DAC output from the microcontroller, which provides a voltage between 0 and 3.3 V. Q507 is a P-channel power MOSFET and the other transistor network controls it as a feedback amplifier, whose gain is controlled by potential divider R508 and R509. (10 + 1.91) / 1.91 = 6.24 so this is the gain of the amplifier.



The measured performance of the amplitude modulator amplifier turns out to be really spectacular, it is extremely linear. In fact within the range of experimental error it is not possible to detect any deviation from a straight line. It really doesn't get better than this!

This is my MAIN point about polar modulation! With an RF transistor Linear PA, the transistor characteristics are curved and it's very difficult to straighten things out. But here we can design an amplifier with multiple transistor feedback circuits to achieve extremely good linearity, because the amplitude modulation only happens at audio frequencies.

The actual SSB generation

The generation of SSB should conceptually be quite simple. "Should be". Of course, it wasn't.

The first step, is to take the audio and somehow turn it into I and Q components. To do this, we use another Hilbert Transform, kind of the reverse of what we do in the Receive signal path. On transmit, the CPU is a lot more busy than in receive, because there is much more to be done. I had to design a shorter Hilbert Transform with 49 taps. This was not found to be the limiting factor for performance.

The diagram (right) is from the 1952 Kahn paper.

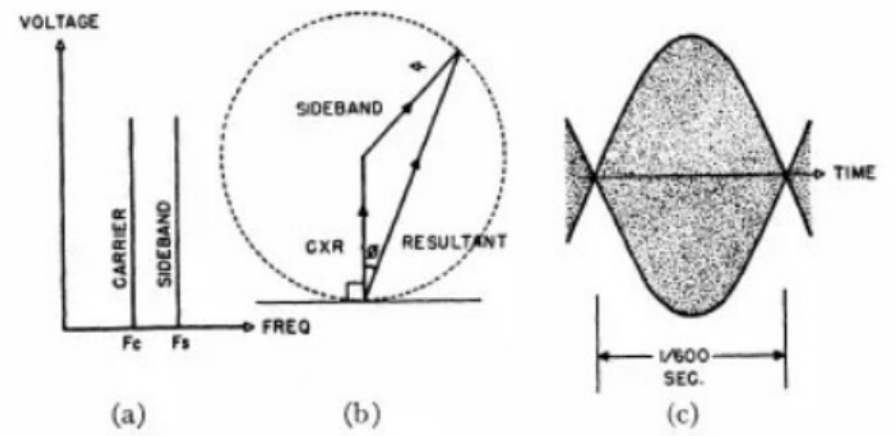
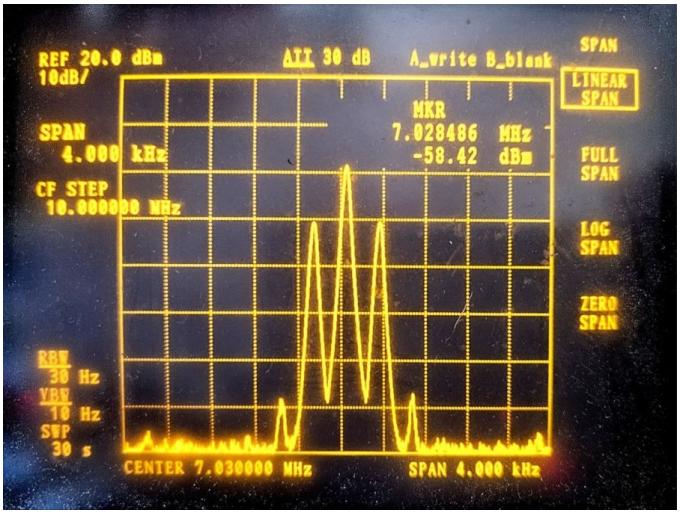


Fig. 1—(a) Spectrum diagram of single-sideband wave. (b) Revolving vector representation of single-sideband wave. (c) Envelope waveform of single-sideband wave.

Once we have done this Hilbert Transform, if we take the arctangent of Q / I, we will end up with an angle. And if we take the magnitude of Q and I - which means, the square root of ($I^2 + Q^2$), then what we have really done, is converted the conventional amplitude vs time representation of our SSB signal, to a polar coordinates system. Each successive sample may be considered to rotate a vector to a new phase angle and a new amplitude, as in Kahn's diagram.

An amplitude modulated signal has a carrier and two sidebands. If we apply a straightforward audio signal to the amplitude modulator, this is exactly what we end up with, see spectrum (right), which shows a single tone modulated by AM. There is a strong central carrier and two sidebands containing the signal information. This signal could be received on any conventional household AM radio receiver.



But an AM signal wastes a lot of bandwidth because the signal information is sent twice (upper and lower sideband); and it wastes a lot of power, both because the signal information is sent twice, AND because of the central carrier which carries no signal intelligence. This is one reason why we radio amateurs love SSB! It makes efficient use of the radio spectrum and it makes efficient use of every watt of power, because it doesn't waste any in the unwanted sideband or the carrier.

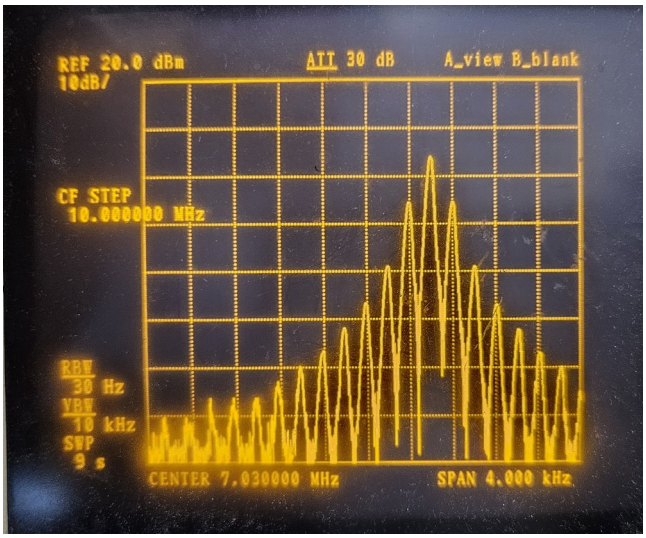
Once we have the polar representation of our SSB signal, we have a phase angle which changes at every sample, and an amplitude which also changes at every sample. It turns out that we can apply these

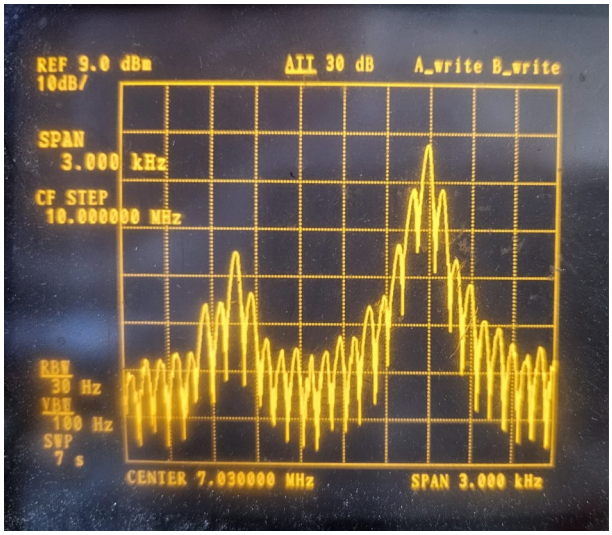
modulations separately: Phase to the MS5351A via rapid frequency updates as described above, and amplitude to the Amplitude Modulator circuit described above. The Power Amplifier recombines everything and if all is well, we end up with only our desired sideband.

So the whole SSB modulation process (of the two tone signal) must consist of:

1. A Hilbert Transform to make the audio signal into an I Q representation
2. Take an arctangent of Q / I to get the phase angle of the polar coordinate signal version
3. Calculate sqrt ($I^2 + Q^2$) which gives us the magnitude of the polar coordinates signal
4. Calculate the frequency shift to apply (to obtain the phase shift between the last calculated sample, and the current one)
5. Send the frequency shift to the MS5351M synthesizer, and the amplitude to the Amplitude Modulator

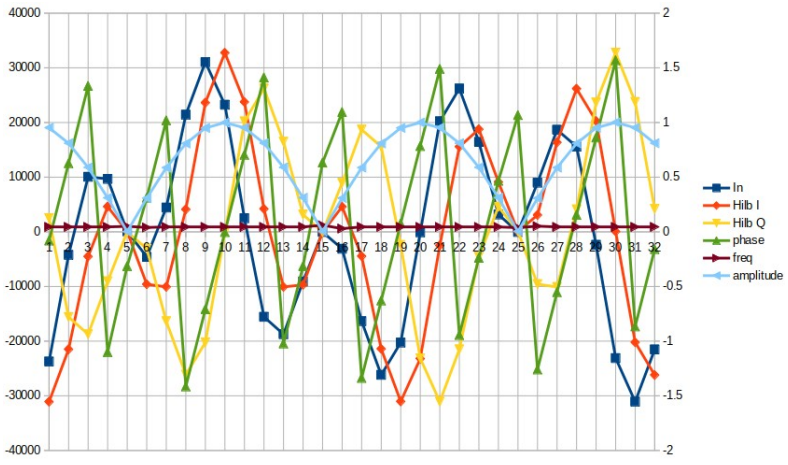
So that's what should happen. I wrote the code, it wasn't complicated, and of course nothing worked. For several torturous WEEKS, nothing worked.





No matter how hard I tried, I could never get those two magical tones to appear on the spectrum analyzer. I tried lots of different things but I always got some variation of the above, which is some kind of AM signal (with various horrendous distortions depending on what nonsense I was trying in my desperation).

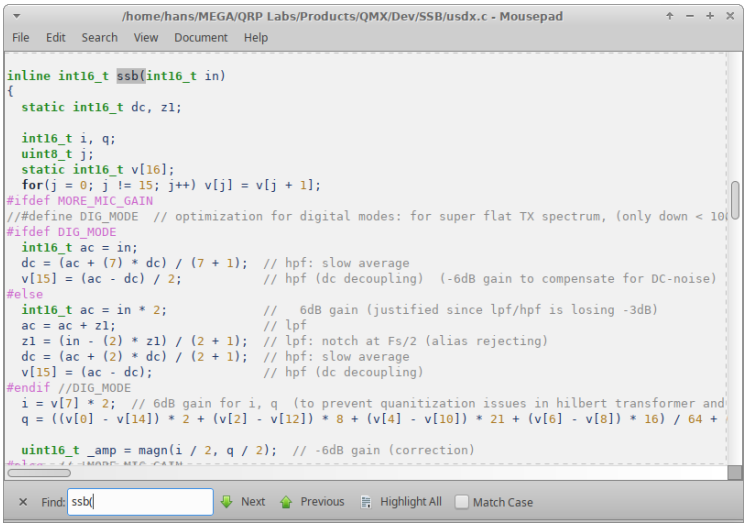
I was losing faith so badly that I was really starting to doubt whether Guido PE1NNZ even really managed to do it at all. Maybe like me, when you were a kid you made one of those 1-transistor FM bug projects. If you yelled into it, then someone with an FM radio in the next room could probably just about hear you. It wasn't transmitting very decent FM, and it could probably have been demodulated on an AM receiver too… but it had enough FM-ism about it that an FM receiver at short range could decode it.



Along the way I even developed a kind of "DSP oscilloscope"! I set up some number arrays in the code and a trigger to take an "oscilloscope screenshot"; at that point the arrays were populated with a number of samples, with columns of the array filled by

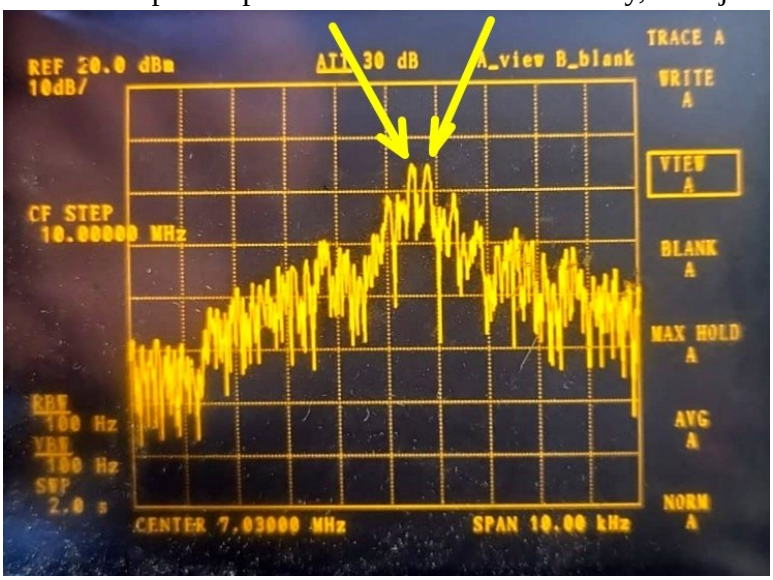
numbers taken from various points in the DSP signal path through the code. It was just like clipping your oscilloscope probe onto various parts of a real analog circuit. Then I had a piece of code to dump the array contents as a comma delimited file to the serial port (QMX has a built-in USB Virtual COM serial port). With PuTTY terminal emulator connected and with logging to file enabled, I could then extract the comma delimited data and plot it in a spreadsheet and plot the “oscilloscope channels”. This was enlightening and solved many problems. But I still could not make it work.

One day I was on the phone complaining to Ross EX0AA about how miserable the whole situation was. Somehow together, we had the idea to try Guido’s actual uSDX code.



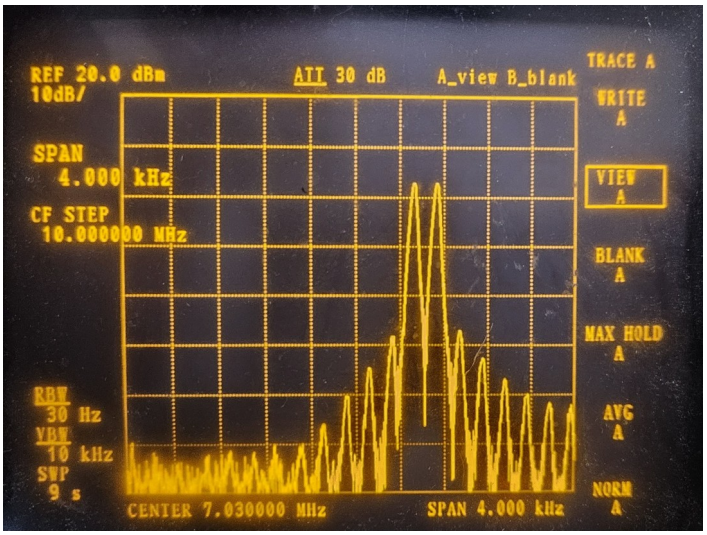
That code is in C, sure it’s written for Arduino and there were lots of differences to my development environment. But I should be able to interface to it without too much trouble.

Guido’s code runs at 4.8 ksps and does everything with 16-bit integers. He uses some approximations for the arctangent, squareroot and Hilbert transforms. because he simply has very little CPU power to play with on that 8-bit ATmega328. So I changed my two-tone code to run at 4.8 ksps and pasted in Guido’s code directly, with just a conversion from floating point audio sample to integer, at the input to his code. I kept my own MS5351M frequency update and DAC amplitude update functions because I had already convinced myself these were working properly.



This historic spectrum analyzer photograph (right) records the moment I was jumping for joy, because for the first time after so many weeks of banging my head against the wall, I was now able to see the proper two peaks sticking out! At that time I was using 600 + 850 Hz tones and the peaks were in the correct place on the spectrum analyzer screen for upper sideband. Amazing!

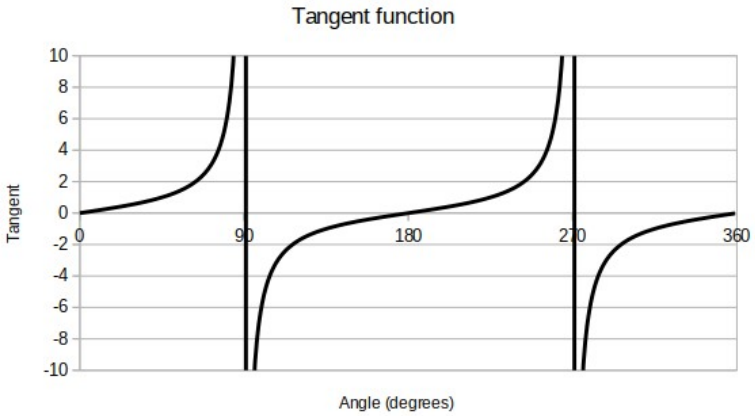
Now there’s a lot of other stuff on the screen, it’s not a particularly clean signal, but we already know that there were many compromises inherent in the uSDX due to the limited hardware and processor capabilities.



So now the question was, why does that code work and mine doesn’t, when mine didn’t use any approximations, I used real Hilbert transforms, squareroot and atan functions. I began to painstakingly replace each line of Guido’s code with my own line of code. After each one line change, I went back and re-measured with the spectrum

analyzer. As I went through my changes, replacing approximations with more exact calculations, the two-tone output signal got cleaner and cleaner.

Until suddenly it went back to Amplitude modulation again… when I replaced the arctangent line of code with my call to atan() in the math.h library!

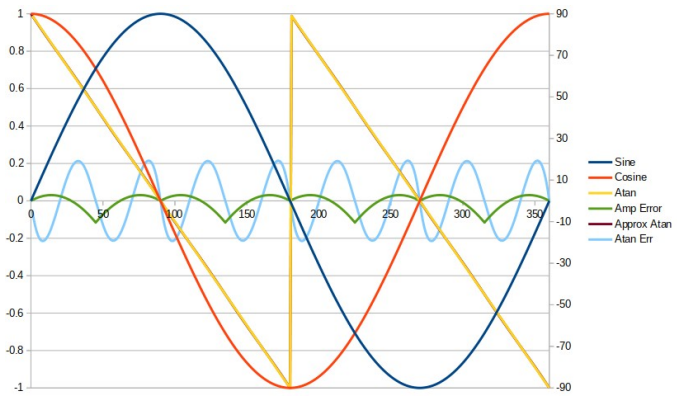


The answer is THIS simple. It's just been a long time since school. The tangent function for 0 to 360 degrees looks like the diagram (right). It repeats TWICE for the range 0 to 360 degrees. The arctangent function, which calculates backward from the tangent to the angle, has no way of knowing whether you meant to be at 90 degrees or 270 degrees (for example); so it always returns a value in the range -90 to +90 degrees. This doesn't then properly cover polar coordinate space, it only covers HALF of the circle, it can't reach the other half. When the signal vector ought to have traveled around the circle into that other half, it really gets "reflected" back into the only half we can access. So what we end up with is AM, no unwanted sideband cancellation or carrier suppression!

Guido's arctangent approximation was taking into account the sign of the I and Q signals, which enabled it to resolve the ambiguity and work out which half of the circle the signal is in. It turns out that the C math.h library has another arctangent function called atan2() which takes TWO parameters (literally exactly our I and Q) and specifically the documentation says it is for conversion to polar coordinates; it looks at the sign of I and Q and returns an angle around the complete circle, in the range -180 to +180 degrees (Note: actually everything is done in units of radians, but this is not important to the concept).

Further improvements

I started increasing the sample rate of the DSP calculations, first to 9.6 ksps then 12 ksps. Each time, the IMD results got better. But it is important in the end, to decide where to spend the CPU power. The power of this 168 MHz 32-bit ARM Cortex M4 CPU with its floating point unit and DSP instructions is amazing. But amazing is not the same as infinite. Particularly when you are doing long Hilbert Transforms, arctangents, square roots, and everything is being done over and over 12,000 times per second! Arctangents and square roots actually do many floating point operations as they are evaluated via a series expansion. So they are relatively slow.

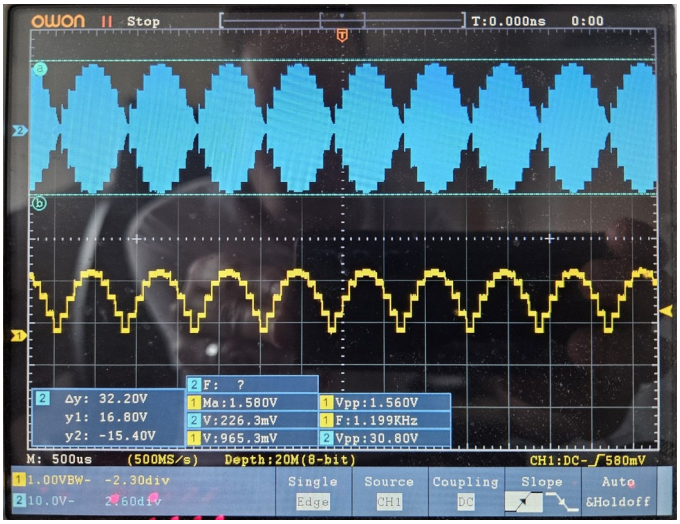


I experimented with changing back and forth to the approximations for the square root, arctangent and Hilbert transforms that Guido had used (in floating point equivalents). From my experimental results I decided that an accurate Hilbert Transform made a big difference, a 49-tap transform fit in the available calculation time and increasing further beyond this did not make much difference to the IMD products. Use of the real square root function (not the approximation) was also critically important. But the arctangent approximation which has an accuracy of about +/- 0.2 degrees was perfectly adequate and much faster than the math.h atan2() function: use of the real atan2() function with its very high precision, did not make much any measurable improvement to the IMD products. I plotted many spreadsheet simulations to investigate the behavior of these functions (see above).

Proper synchronization between the phase and amplitude modulation is an absolutely critical matter. If the phase modulation and amplitude modulation hit the power amplifier at slightly offset times, the intermodulation performance is impaired. It takes some considerable time for the phase (frequency) update to be written to the MS5351M via the I2C bus. In fact as the maximum update rate to the MS5351M with a 1 MHz I2C bus speed is 16 ksps, you can see that at 12 ksps the system requires approximately 75% of the sample time to write the new frequency command. Even then, the phase shift is the integration of the frequency shift over time (see previous description), so the phase does not reach the actual commanded phase until a whole sample later. So how, and at what point, should we try to delay the amplitude modulation to match the result of the phase modulation? This question and how to calibrate it, became a critical matter, which is described below.

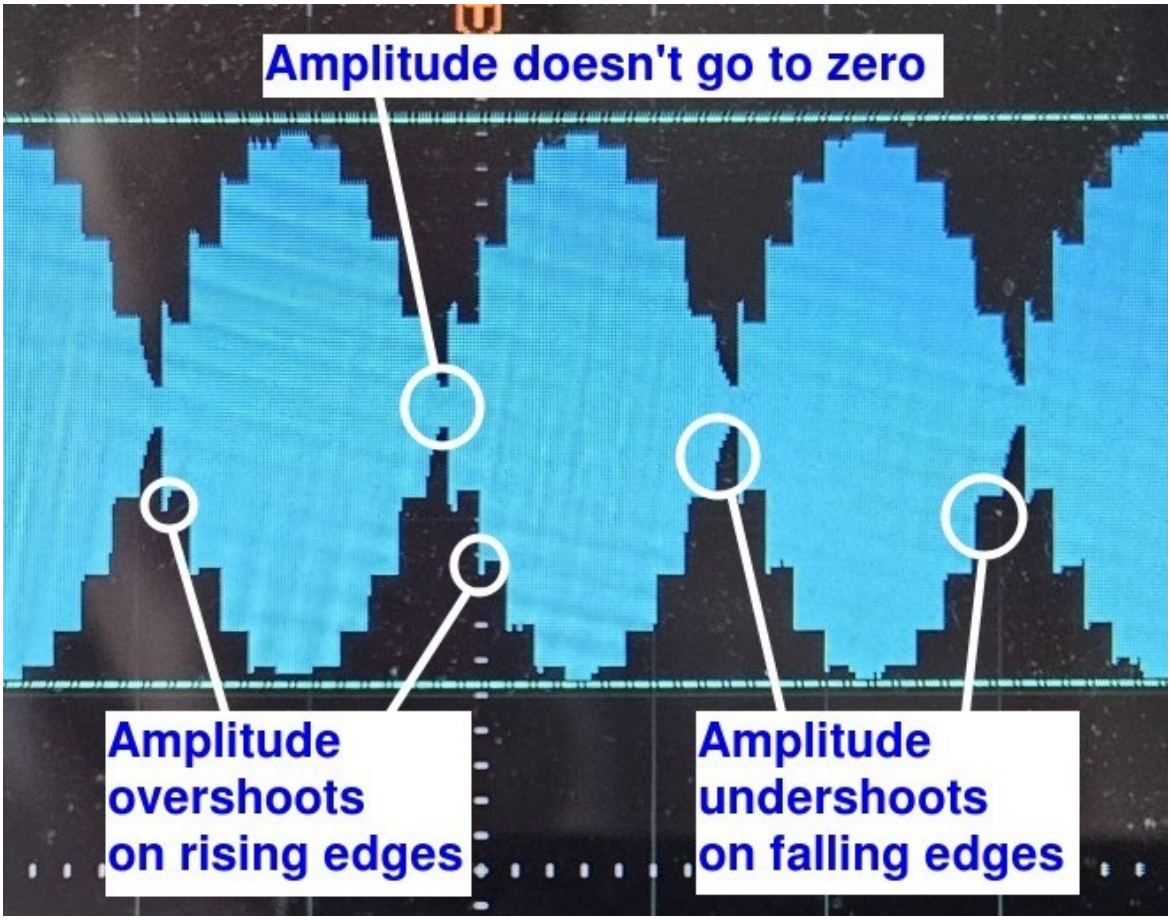
Over-shoot / undershoot and the interpolation solution

There was a big problem. The oscilloscope photograph (right) shows a 700 Hz + 1900 Hz two-tone modulation. The bottom trace is the DAC output to the amplifier and is perfectly fine. The top trace is the RF output across a dummy load.



These peaks, like a rectified sinewave, occur at a frequency of 1200 Hz. This amplitude envelope modulation occurs at the frequency difference between the two tones.

You can also see that there are 10 steps per cycle, this is because the DSP processing is running at 12 ksps (and 12,000 / 1,200 = 10).

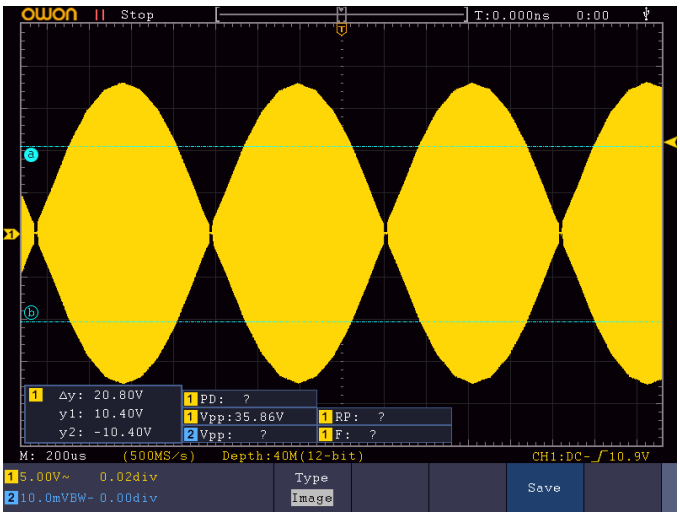


The issue is labeled on a zoom of that photo (above). The problem is that on the rising edge of the waveform, on these rising steps, the amplitude modulator “overshoots”: it momentarily passes the desired step value. On the falling edge, the amplitude “undershoots”: it seems to take some time to decay down to the wanted value. Another feature (which is dealt with in the next section) is that the amplitude never goes totally to zero, either.

By this stage in the project I had started to build up quite a power of intuition about the whole thing and I surmised – correctly, it turned out – that this overshoot/undershoot would be significantly harming the intermodulation performance. I imagined that if I was to interpolate the amplitude samples and send them at a faster rate to the amplitude modulator, then the little steps to the amplitude waveform would be a lot smaller, and the overshoot distortion would also be a lot smaller.

In this photograph (right) I interpolated by a factor of 8, with a straight-line linear interpolation, and you can see the huge difference it makes! Now the amplitude samples are sent to the DAC at a rate of 96 ksps. If you zoom in (see bottom right of the image) then you can still see the overshoots; but now the steps are so much smaller, and the overshoots are correspondingly much smaller too. I measured an immediate improvement in the level of the intermodulation products on the spectrum analyzer.

I tried even higher interpolation levels, at 16x, 32x and 64x. As I went to 16x and 32x, the performance still measurably improved, but by a lesser and lesser amount of improvement. At 64x the whole thing just ground to a halt and the software crashed! So I left that as it was, at 32x interpolation.



In the end, 32x interpolation does not work out as a tidy division of the 168 MHz system clock, and did not fit well with the microphone sample rate, without leaving fractional divisions and missed samples. So I changed it to 28x and everything fit perfectly.

In this image of the two-tone test, the linear interpolation factor is 28x, meaning an amplitude sample rate of 336 ksps. There is NO visible trace of any overshoot/undershoot. So the interpolation trick neatly solved this deficiency in the amplitude modulator performance.

Note that I did try other more elaborate forms of interpolation involving polynomial splines, and fitting sinewaves to implement a sinewave form of interpolation. These hurt my head, but they did work, and there was some minor improvement in performance compared to the straight line linear interpolation. However, they consumed a totally disproportionate amount of CPU resources, which if spent elsewhere would result in a more easily obtained performance improvement. The complicated forms of interpolation were therefore dropped in favor of the quick and easy, linear type.

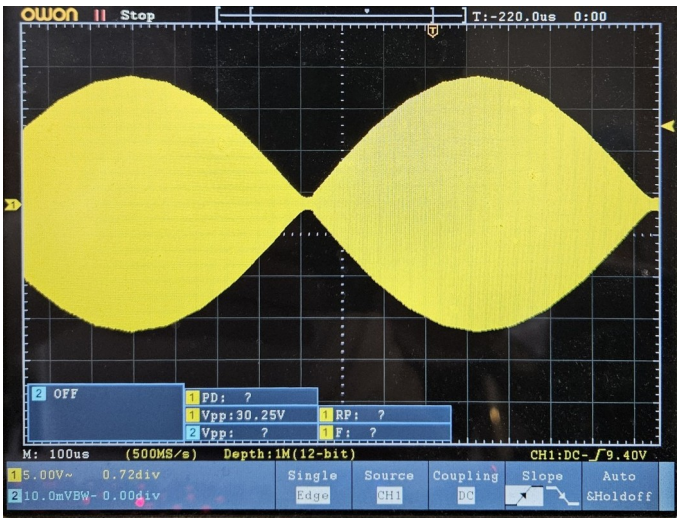
There are two more critically important side effects of this linear interpolation of amplitude samples thing.

Firstly, now there are 28 sub-samples, interpolated between each sample that is calculated by our DSP processing output. This makes it very easy to set up a digital delay line of the amplitude sub-samples, and arrange a variable offset for achieving synchronization between the phase and amplitude signal paths. A

digital delay line of this nature is easily implemented in code as a circular buffer, certainly a much easier thing to do in the digital world than it would be in analog electronics! So this interpolation of amplitude samples rather neatly solved the problem of how to obtain a synchronization resolution smaller than one 12 ksps DSP sample. It doesn't solve the problem of WHAT the synchronization delay should be, but it does at least allow us to implement one.

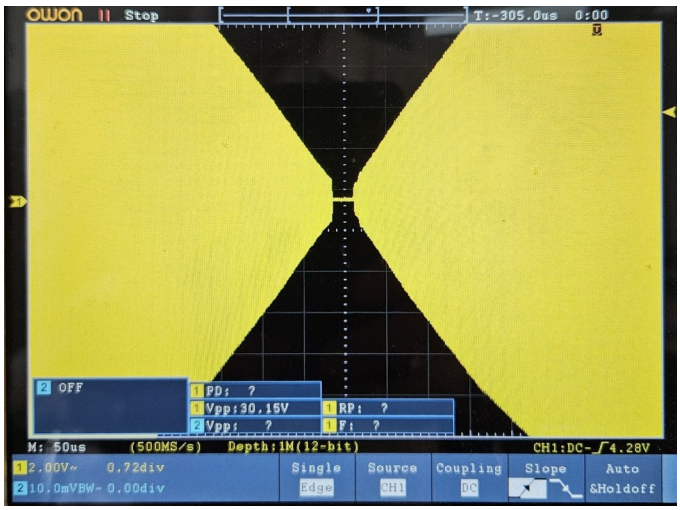
The second very important outcome of doing the straight line linear amplitude interpolation between DSP calculated samples which run at 12 ksps, is that it now much more closely approximates what happens to the phase modulation. Remember that the phase shift is implemented not by real phase changes, but by carefully calculated frequency changes. The phase change is the integration over time from one sample to the next. In other words, the phase slides in a linear straight line, from one calculated sample value to the next! If I do 28x linear interpolation of amplitude samples, then they'll also be close to a straight line; so we can speculate that the overall effect is somewhat similar to a system running much faster than the true 12 ksps. I believe that this is the reason why the measured IMD3 performance is much better than the -30 dBc rule of thumb that was mentioned previously: because the 1200 Hz tone separation is 1/5th the DSP, amplitude and phase modulation bandwidth of 6kHz, the Nyquist frequency of the 12 ksps sample rate. And the rule of thumb is 5x. It seems likely to me that this "smoothing" effect on both the amplitude and phase modulation, and them closely tracking each other, lets us significantly defeat this rule of thumb.

Amplitude pre-distortion



The other problem I mentioned was that the amplitude never quite goes to zero at the minimum (zero) point of the two-tone amplitude envelope. The reason for this is a limitation in the design of the QMX. The amplitude modulator supply voltage to the PA goes close to zero. But, the driver (a 74ACT08 logic gate chip) is still hitting the MOSFET PA transistor gates with a 5V squarewave. The transistors are imperfect too (like any other analog component) and have a reverse capacitance between their Drain and Gate. This allows signal to leak through the MOSFET, even when the DC drain voltage is zero.

At first sight, there's not a lot I could do about this. An absolutely inviolate constraint of this SSB project is that it should require solely firmware updates, no modifications to the hardware. Because there were already ~9,000 QMX/QMX+ out in the field by the time the SSB firmware was ready. Many purchased as pre-assembled and tested units from QRP Labs. We can't expect all those thousands of people to start making modifications to the surface mount PCB!



So instead I came up with a solution, to blank the signal completely when it falls below a certain level. This is shown switched on in this oscilloscope image. It's a sudden cut of the signal when the amplitude gets below about 1 V. I didn't really consider this to be an ideal solution but had no other options.

Measurement of the IMD spectrum does show a marginal improvement with this feature enabled. The IMD3 product drops about 1dB. Higher order distortion products have a varied effect, some

increase, some decrease; but overall when you stare for long enough at the before and after spectrum analyzer photographs it does look like there is some tangible benefit. Since the benefit is small, and I can not be certain that there would be any unpleasant side effects in other scenarios than the highly artificial two-tone test (though I have never observed any problems), I left this feature as a configuration parameter under the name Amplitude pre-distortion.

Conventional transceivers by big name manufacturers like Icom, Yaesu etc often advertise that their transceivers implement DPD: Digital Pre-Distortion. This is a form of compensation to cancel out the non-linearities in the RF power amplifiers. In the case of the QMX, the amplitude modulation occurs at audio frequencies and as I described earlier, the thing is so linear that I could not measure any non-linearity at all outside normal measurement error. So in QMX the only thing “Amplitude pre-distortion” does is switch on or off this transmit signal cutting effect at low amplitudes.

It should be noted that this is very much a perfectionism issue; the 1Vpp signal which is not cut is 2.5 milliwatts, 37 dB below a 5W PEP signal. So the levels are very low and arguably of very little importance.

Phase pre-distortion and calibration

Now we come to the topic of phase pre-distortion which is a really big and important one. It’s one thing carefully calculating the phase angle of the polar coordinate space SSB signal and modulating it carefully as a series of frequency updates to the MS5351M synthesizer. But if the Power Amplifier itself has phase distortion then all that care and attention is wasted.

It turns out that there IS significant phase distortion and compensating for it, does make a big difference. The phase distortion is manifested by a variable phase delay through the power amplifier, depending on the amplitude of the signal. One can imagine that perhaps the MOSFET body diode behaves a little like a varactor diode, varying its capacitance as the applied voltage changes.

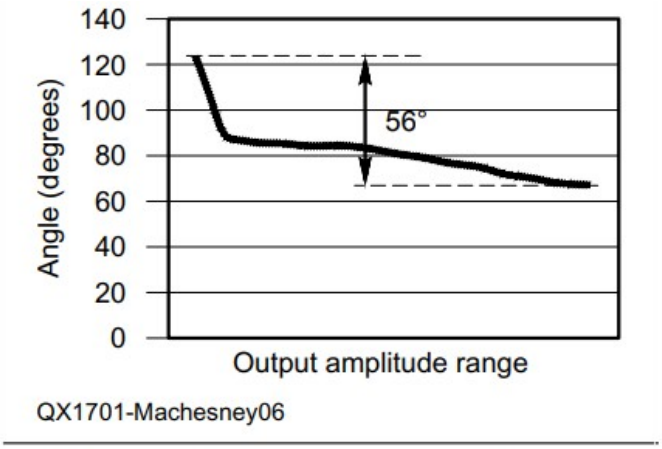


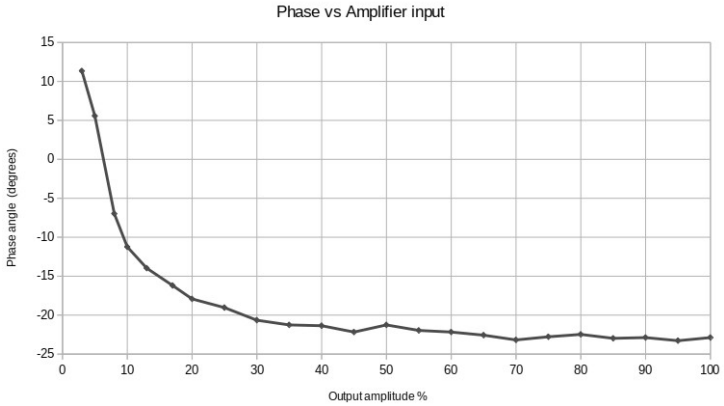
Figure 6 — The uncompensated power amplifier input vs. output phase varies by 56 degrees across the output range.

The 2017 QEX Polar Explorer article (referenced earlier) also noted the same issue with phase distortion and in their case, also amplitude distortion. The designers solved it by the use of an Analog Devices AD8302 gain and phase detector chip, which they use to build up a static table of the phase and amplitude distortion, which they then applied as pre-distortion coefficients to cancel out the undesirable distortion.

I looked up the AD8302 at Digikey and even in 2500 pcs quantity on a reel, the price is still almost $19 a piece! I was shocked by the price, but anyway as I mentioned previously, a rule or “constraint” of the project is that it must require NO hardware changes for existing QMX owners! Which rules out any AD8302 chip.

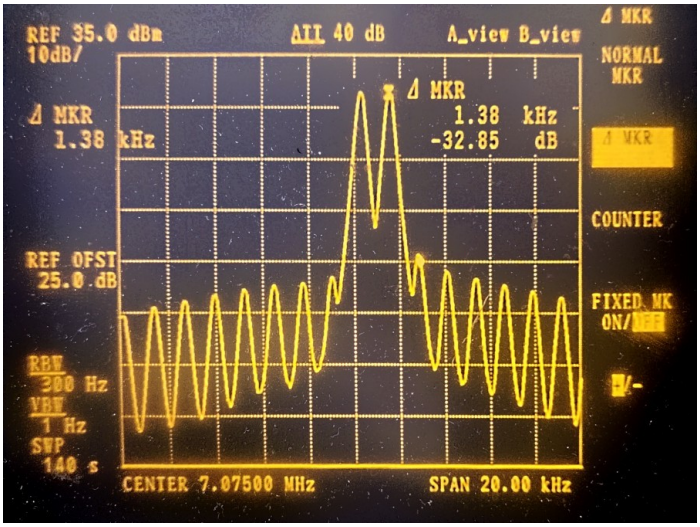
So I set up a measurement of phase error, which I unfortunately did not record photographically, but is easy to describe. I simply connected one channel of my oscilloscope to the 5V squarewave at one of the gates of the BS170 PA MOSFETs; and the other oscilloscope channel to the output of the whole QMX at the antenna port. I transmitted with a simple pure carrier of variable amplitude. Then I could take an image of it on the oscilloscope use the oscilloscope cursors to measure the time difference between zero crossings, and convert that to a phase delay.

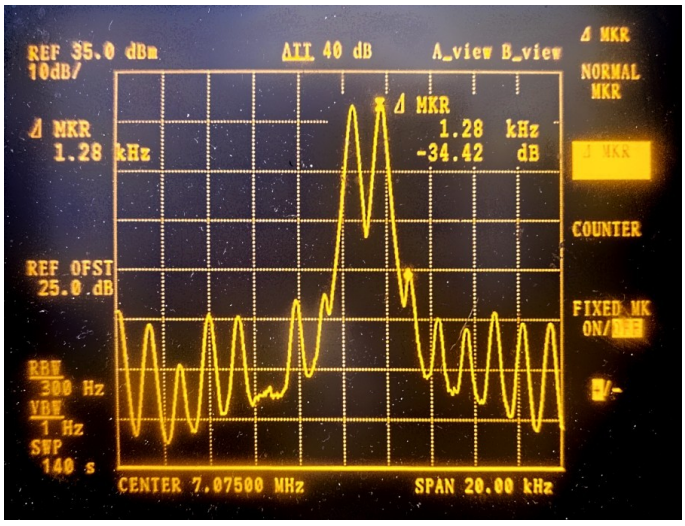
This measured the phase delay between the MOSFET gate and the transmitter output. That is the summation of phase delays due to multiple factors, some of which are fixed and only one of which will be distortions that depend on the amplitude. But the absolute phase delay does not matter! All we care about is the change in phase relative to applied amplitude (voltage). Any absolute fixed phase delay will already be accounted for once we adjust the phase vs amplitude modulation synchronization.



The result was a graph (right) of phase delay (phase distortion) vs amplitude. Most encouragingly, the shape was quite similar to what the Polar Explorer dudes found for their transmitter! I coded this into a lookup table and applied its opposite, to the phase samples in the SSB modulator code. So for each sample, which as an amplitude and a phase, I would look up the phase error on the above curve for the amplitude of the sample, then subtract it from the phase error.

The results of applying this phase pre-distortion are a quite notable improvement in intermodulation products. The plot on the left is without applying phase pre-distortion (OFF), the plot on the right is with pre-distortion applied (ON). As well as almost a 2dB improvement in the IMD3 level, there is a much larger drop in many of the higher order intermodulation products.



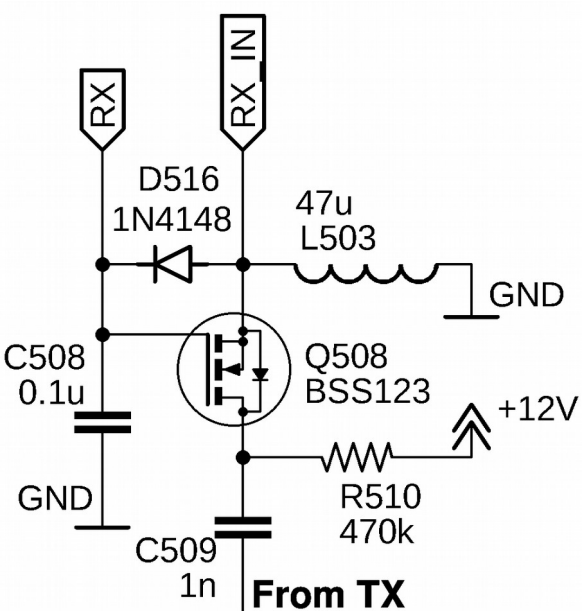


The question then is, how to calculate the pre-distortion table. In the last resort I could make a careful oscilloscope measurement for each band, and hard-code it into the source code of the firmware. However, it is very very difficult to make accurate oscilloscope measurements of this phase distortion by hand on the higher bands. AND, that would be the phase distortion curves for MY QMX. What about other people's QMXs? What if they wound their toroids differently, or the BS170 transistor characteristics were a little different, or, or, or?

But how could I make a phase measurement, without the use of an expensive chip like that AD8302, and even more importantly, without making any hardware modifications? I was mulling this over in the back of my mind for several months, intuitively feeling or perhaps just WISHING, that there must be a way. Eventually it came time in the project, to sit down and try to make it so. I devised the following solution.

There is nothing to stop me from using the QMX receiver, at the same time as it is transmitting on its transmitter. Particularly if I am just transmitting an unmodulated carrier, not messing around with SSB modulation, it's very easy and requires no CPU resources, so there is plenty of time left over for reception.

The QMX has a simple Transmit/Receive switch, shown right. In TX this transistor is OFF and blocks the large amplitude signal passing through to the receiver circuits.



But as I keep saying, nothing is perfect! So something will leak through this circuit regardless. Which means that we can detect it using the receiver.

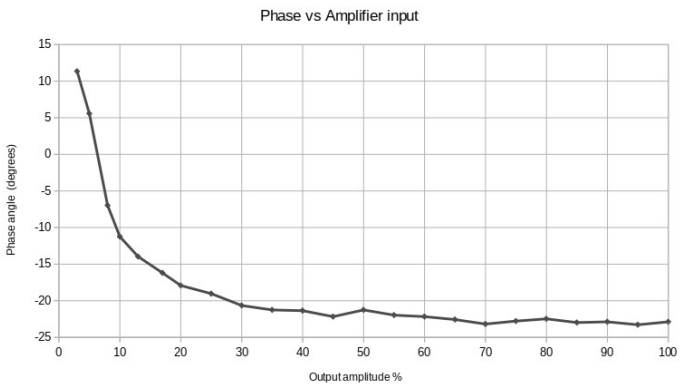
Now imagine if we start off with the Receiver Local Oscillator active and set to EXACTLY the same frequency as our transmit signal. What do we get in the receiver? Nothing at all, we're at zero beat! You simple homebrew TX CW operators know about this. There might be some DC offset level but the PCM1804 has a High Pass Filter with a 1 Hz cutoff inside it. So what we detect in the sample stream coming from the PCM1804, is zero, nothing.

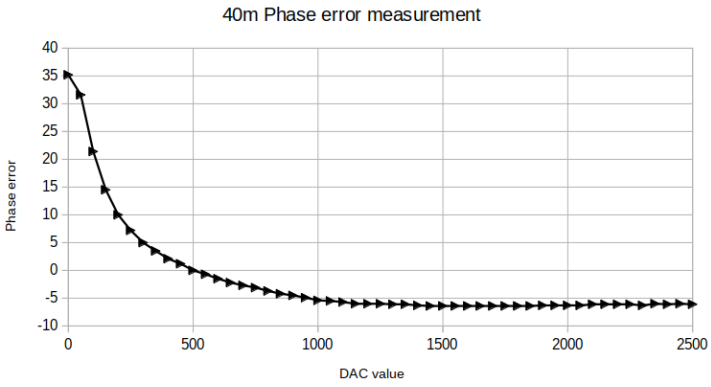
Next, at a certain exact time, we do two things. Start a timer, and move the transmit (or receive) frequency by a small amount, say 5 Hz. I mean really 5Hz, nearly EXACTLY 5Hz to a few micro-Hz of precision. Now what is heard in the receiver? A 5 Hz tone! But of course, there's a 1 Hz HPF and there is (or was) a big DC offset somehow, so there will be an initial big DC offset to the waveform and we will need to wait a little time for it to settle down.

After waiting a suitable time, now the 5 Hz audio tone has no DC component, it is centered perfectly on zero. And we can carefully watch the samples coming in, and determine the exact moment of zero crossing from negative to positive, which will indicate the completion of one audio cycle at 5 Hz.

Now the important point is this. We KNOW what time we expect the zero crossing, because we are running a timer that we started at the exact instant that the frequency offset was changed to 5 Hz. And we can measure the actual time that we see the zero crossing occur. The difference between these two, is a time – and we know the period of the sinewave (0.2 seconds, for 5 Hz); so now we can convert this time difference to the phase angle (360 degrees * time difference / period). This result IS the phase delay through the power amplifier. Everything is extremely accurate, the timing to microsecond precision, frequency to micro-Hz precision, etc.

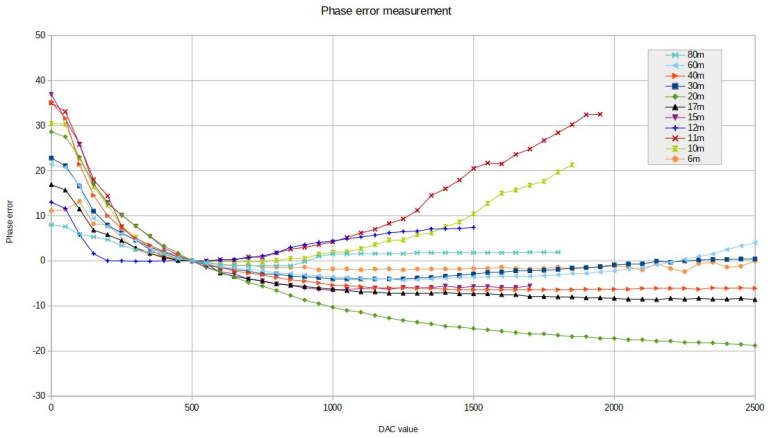
Again: there will be plenty of unknown phase delays through the system. Not just in the power amplifier, but also in the Tx/Rx switch, the band pass filters, ADC, sample processing, etc. We don't know any of these. But if we can reasonably assume that they are fixed, then the only variable is the phase distortion vs amplitude. Fixed delay will be taken care of by the phase vs amplitude modulation synchronization.





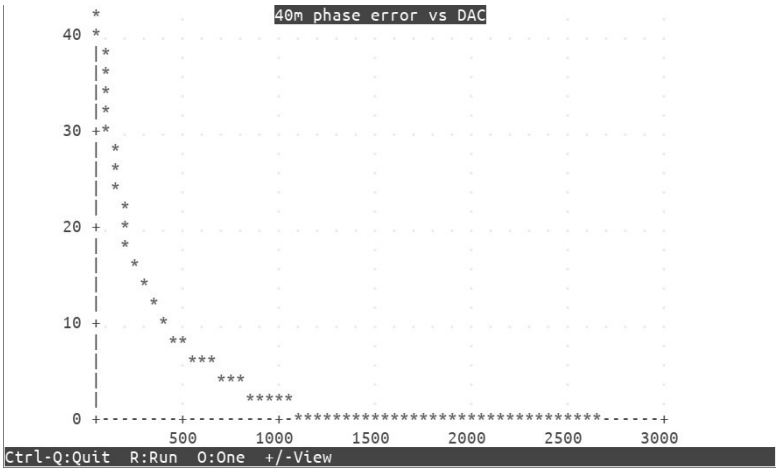
Now here I put the two measurement results side-by-side, with the manual measurement on the left and my self-measurement on the right, you can see they aren't a million miles (err, I mean degrees) different, are they! So evidently the method appears to agree with my painful manual method by the oscillscope.

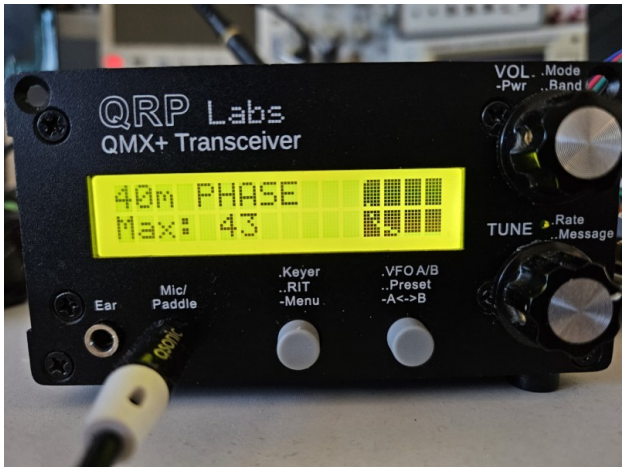
I was able to obtain a plot of phase distortion (phase delay vs amplitude) for every band in the QMX+.



In some cases the amplitude of the leaked signal through the Tx/Rx switch was too high and clipped into distortion before the maximum amplitude part of the waveform was reached.

Having developed a good method for self-measurement of the QMX transmitter's own phase distortion, the next step was to wrap it all up in a nice tool which all QMX owners could run themselves if they wish. So there is a tool when you log into QMX with the serial terminal, that allows you to run the calibration. It's quite slow because it has to wait on each step, for the PCM1804 High Pass Filter to get rid of the DC, and as it's a 1 Hz filter, this takes a little time. It steps up the amplitude from DAC 50 in steps of 50, until the supply voltage is reached. Of course it has to do this for each band, too.

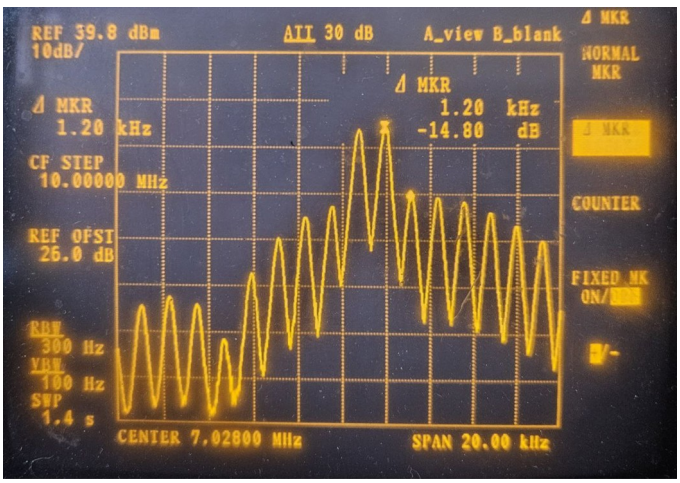




The calibration tool plots the curve in ASCII art when logged in using a terminal emulator; but you can also do it on the front panel LCD itself, accepting the very low resolution of the available custom pixel graphics of the alphanumeric display.

Synchronization and calibration

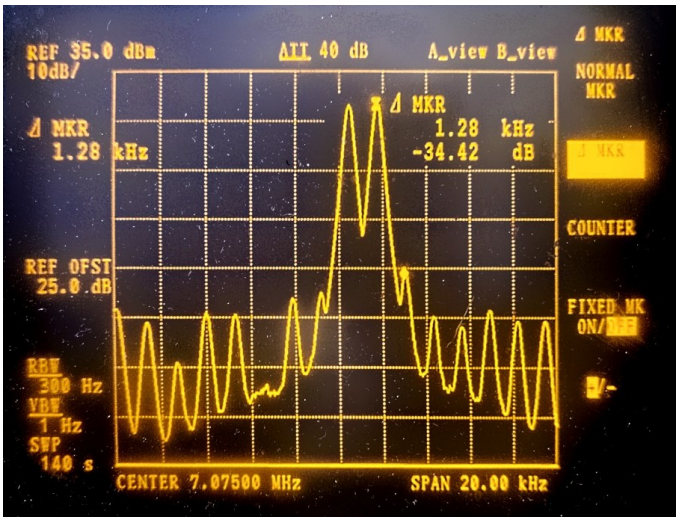
Eventually we come to the tricky topic of how to correctly synchronize the phase and amplitude modulation such that they both effectively arrive at the same time at the PA output and beautiful SSB is the result. There could be variation per band, on upper vs lower sideband, and it will depend somewhat on the phase pre-distortion too, as well as build and component variances from one transceiver to the next. So a fixed synchronization is definitely NOT the ideal solution.



Here on the right, I show an example of the intermodulation products when the synchronization is

set up very poorly. You can still see the two tones but the IMD3 product is only 15 dB down from the carrier (-21 dB PEP). Not a great number.

When the synchronization is correctly set up (by manual adjustment), I was more used to seeing IMD3 around -41 dB PEP on 40m as in this photo (right).



So the crucial question became, could I find a way to let the QMX self-calibrate itself, to find the best synchronization offset for the delay in amplitude samples? Recall that because of the 28x linear interpolation of amplitude samples, we can offset the amplitude in discrete delay steps of 1/28'th of a 12 ksps sample time. So the mechanism for the delay is there, it's now a matter of calibrating how much the delay should optimally be.

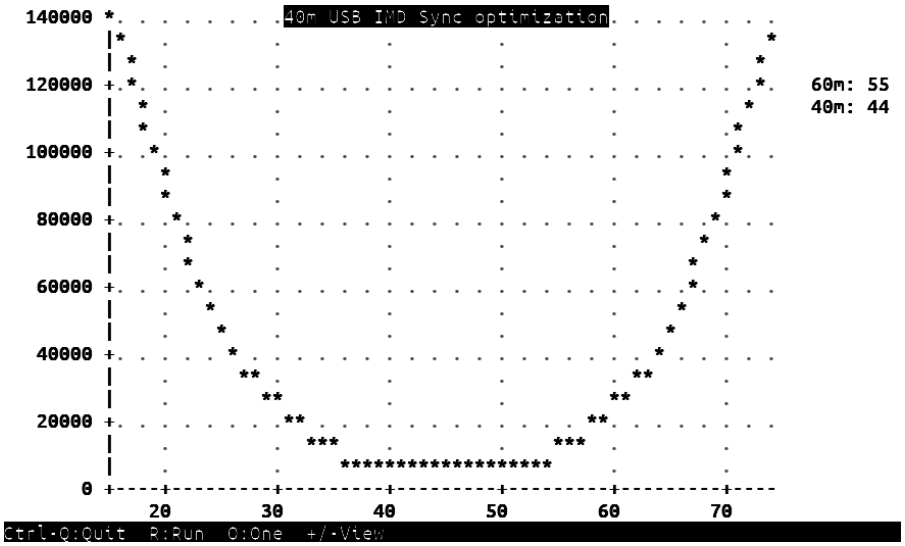
Again, the fact that the receiver can be operated alongside the transmitter at the same time, with some large attenuation occurring through the "OFF" Transmit/Receive switch, is the way to achieve a self-calibration.

I tried a great many different ways to measure the level of the IMD3 (and higher order) products involving Goertzel algorithm buckets placed on the frequency at which they would occur. In the end I kept coming back to a reliable, stable, easy (relatively speaking) method, as follows.

Firstly, the two peaks are 700 Hz and 1900 Hz. The 3rd order, 5th order and 7th order intermodulation products (assuming Upper sideband mode) on the high side occur at 3100 Hz, 4300 Hz and 5500 Hz respectively (they are spaced regularly every 1200 Hz). Now if I shift up my Receiver local oscillator by an offset of 2700 Hz, the 3rd, 5th and 6th order IMD products will now show up at 400 Hz, 1600 Hz and 2800 Hz. All three are comfortably within the receiver passband in its widest setting, which is 150 Hz to 3200 Hz. As for all the other tests, a dummy load is connected to the transceiver so we can assume that no other signals are present except some noise at low levels.

Now the strategy is to run the SSB receiver, and collect 12,000 audio samples (at 12 ksps, this takes 1 second). For each of those samples I square it (multiply it by itself) The reason for doing this is that these samples are voltage, but I want power. To convert from voltage to power, we have to square the voltage (and then divide by the resistance etc). We don't care about the absolute value, just having a value which is proportional to power, is the desired outcome. This number is then a measure of the total power in the 3rd, 5th and 7th intermodulation products.

Then we can ramp the synchronization delay from 15 sub-samples all the way up to 74 amplitude sub-samples (remember one 12 ksps sample time period contains 28 sub-samples due to the 28x linear interpolation). At each point, we



wait 1 second while the power data accumulates. That's then plotted on an ASCII graph (or on the LCD if you run it from there!), and at the end the system can easily determine the synchronization offset which produced the minimum intermodulation energy.

So that quite neatly solves the optimization of the synchronization parameter. The calibration tool runs this per band, and again for lower sideband.

Microphone sound handling

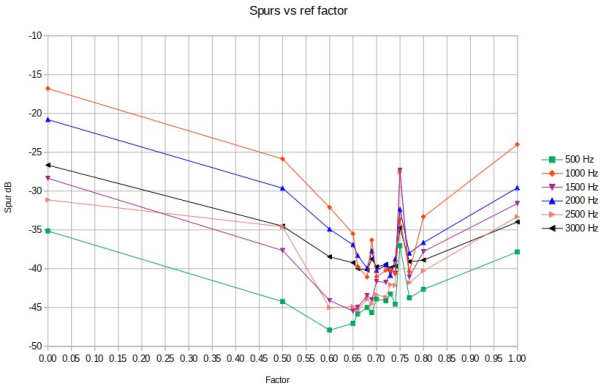
The QMX has an internal onboard microphone and an external one can be plugged in to the paddle port. At the time of writing only external microphones are supported. This section describes the processing specific to the microphone samples.

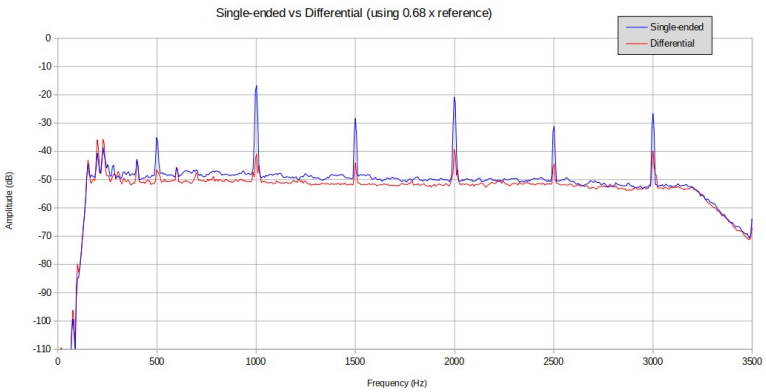
Sampling and noise

The QMX microphone is, for better or for worse, connected directly to an ADC input pin of the STM32 microcontroller, which has three 12-bit mutiplexed ADC peripherals onboard. In order to gain additional microphone sensitivity, the microphone is sampled at a rate of 700 ksps (700,000 samples per second). Downsampling by a factor of 58.333 (700 ksps / 12 ksps) significantly improves sensitivity.

Unfortunately despite the impressive 12-bit resolution of the microcontroller ADC peripheral, these ADC's were found to be rather noisy. Spurs are found to occur every 500Hz across the audio spectrum. It was found that the noise is highly correlated between different input channels on the same ADC peripheral. Therefore a reference channel is sampled, also at 700 ksps and also downsampled by a factor of 58.33. If noise cancellation is switched on, the reference channel noise signal is subtracted from the microphone signal path, substantially canceling the noise on the microphone signal.

In the images below, on the left I show the spur measurements vs the factor of reference channel subtracted. On the right, the audio spectrum plot before (blue, no reference channel subtraction) and after (red, with 0.68 x reference channel subtraction). The audio spectrum was made by recording 30 seconds of silence in a quiet room, streamed to the QMX USB sound card and recorded on the PC in Audacity, which then did the spectrum analysis. The worst spur at 1kHz is reduced by about 24dB and other lower spurs are also significantly reduced. Practically the result is that without noise cancellation, these spurious tones are head clearly at low level as background noise under the speech. When noise cancellation is on, no spurious tones are audible at all.





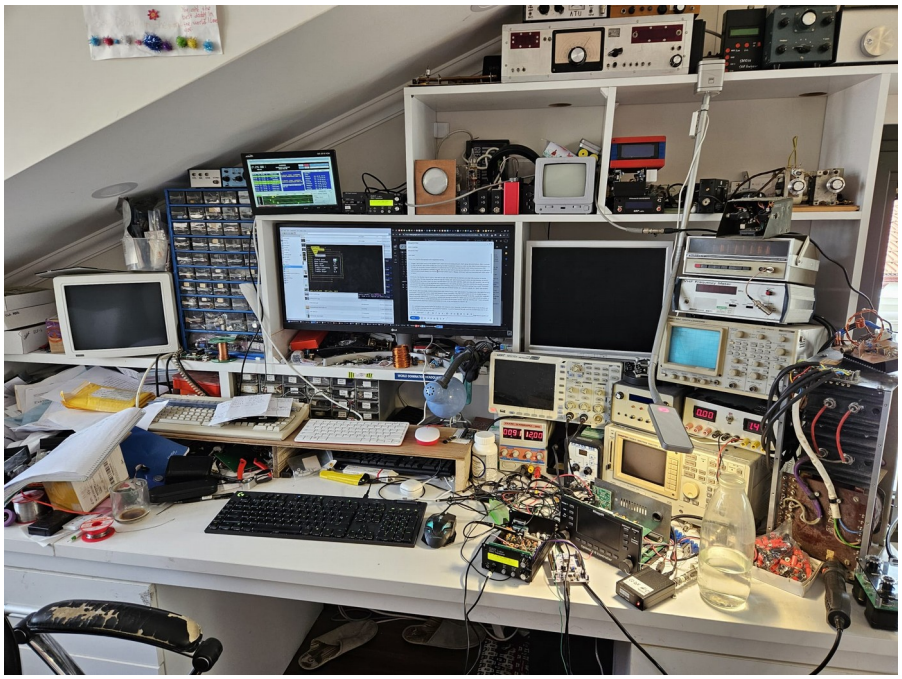
Mic noise gate

A minor issue is the leakage through the PA even when the amplitude voltage is zero, which I mentioned previously in connection with amplitude pre-distortion (see above). This occurs because in the QMX circuit, the BS170 PA transistor gates are driven with a 5V square wave from 74ACT08 logic gates. The drive signal

is present on the BS170 gates even when the Drain voltage is zero (zero amplitude modulation). Some of this signal leaks through the PA transistors in this OFF condition due to their capacitance.

Practically speaking, the Peak-Peak RF output of the PA is extremely linear with respect to the control voltage from the microcontroller's Digital to Analog Converter (DAC) output, from 45 or 50V peak-to-peak (5W is 45Vpp) down to below 1V. But somewhere between 0.5V and 1.0V, even though the DAC value goes to zero, the RF peak-to-peak does not decrease further. The control range is around 37dB of linear straight line control range. Which is good, but not infinite.

Even when the signal coming from the microphone is extremely low, the SSB modulator (polar modulation technique) still tries to convert it to polar coordinate space (angle and magnitude) for transmission as separate phase and amplitude modulation components. Low level microphone noise translates to phase angles that jump around randomly all over the circle. If the amplitude were truly zero, this would not matter. However, as it is, even with 1Vpp you can hear this noise.



This *may* be being FAR too perfectionist. 1Vpp into a 50-ohm system is 2.5 milliWatts. 37dB down from Peak Envelope Power. It was audible to me during testing because I transmitted at 5W PEP from a QMX+ into 79dB of inline BNC attenuators, straight into a QDX acting as the receiver. At the QDX input that results in an S9 + 30dB signal! Meanwhile there is no band noise, because the QDX (as receiver) RF port is piped straight to the QMX+ (as transmitter) RF port via BNC attenuators. 37dB down from Peak Envelope Power still leaves you with an S8 signal, and with a very low noise floor any defects are therefore extremely clear! It's a very very harsh test environment. In real life conditions you would be unlikely to be putting such an enormous signal into the other station's receiver, and he would not have zero band noise and zero other QRM. So these 37dB (6 S-points) down defects may not be noticed.

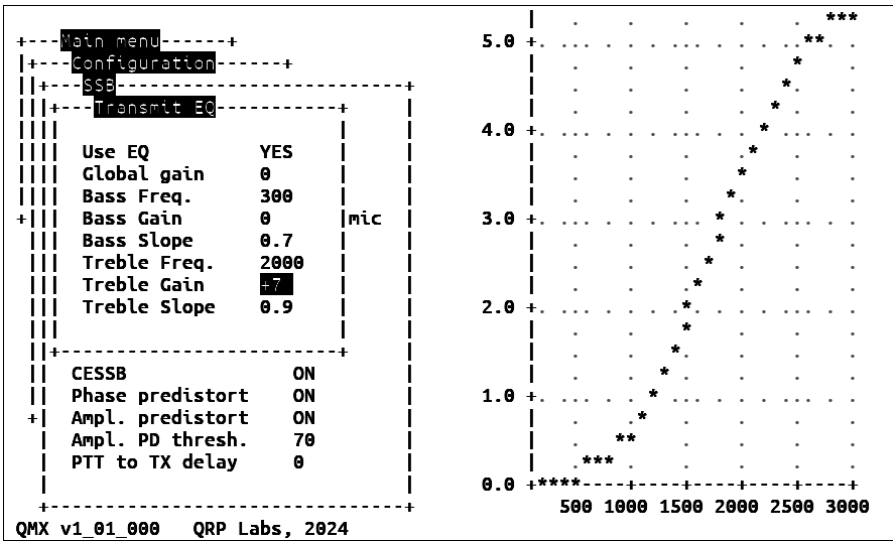
Nevertheless, there’s a facility to gate this microphone noise below a chosen threshold. The gate operates by switching off the TX signal to the driver altogether. It needs to be used with care, since there is nothing between zero output and 1Vpp. If the gate opens and closes too often, it produces a scratchy type of sound at low level on the audio, which is quite annoying.

VOX

VOX means Voice Operating Exchange or more commonly, Voice Operated Transmission. Instead of waiting for a PTT button to be pressed to initiate transmission, the radio monitors the microphone continuously during receive, and if there is a noise it enables the transmitter. In the QMX the audio source can be the USB sound card, so the VOX settings also apply to the USB sound card audio, if USB is selected as the Input source for the SSB transmitter. The VOX subsystem operates in parallel with the PTT button, and indeed the CAT commands. So you can still activate transmission by the PTT button for example, even when VOX is on.

Equalization

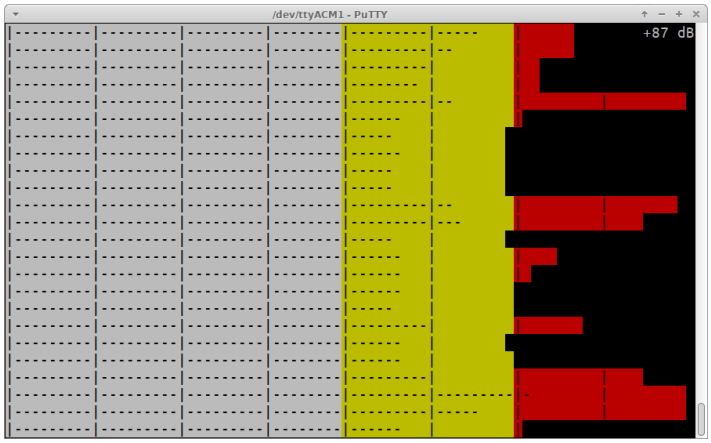
I am not an SSB operator myself. But I note that many DX experts say that a large amount of energy is concentrated in the bass parts of human speech, but this contributes very little to the intelligibility of the transmission. It is therefore recommended to boost the treble end of the speech spectrum and attenuate the bass end. To this end the QMX SSB firmware includes a parametric transmit equalizer with two filters for bass and treble response.



These filters are another type of DSP transform called a Biquad shelf filter. There are configuration settings which determine the parameters used to calculate the shelf filter coefficients. There's a visualization of the response of the filter, which is updated almost in real time around one second after a parameter is changed. The parameters can also be edited on the LCD, and a miniature version of the response curve is plotted on the right side of the display.

Microphone AGC

There is a microphone gain setting and a tool to visualize and test the microphone gain (on both the terminal login and the LCD).



But additional to this tool, there’s also a configurable AGC which can compensate for variations in your speech volume or how close to the microphone you are. It’s kind of a simplified version of the AGC used in the QMX receiver. A configurable amount of gain can be added, with a configurable hang time and recovery rate. This provides quite an effective gain control mechanism even though it consumes very little processing power.

Compression

A configurable gain factor is applied which increases the average to peak power of the SSB transmission, giving you extra punch in weak signal conditions, DX, contests etc. It is applied to the softer quieter parts of the human speech to make them louder. The loudest peaks are untouched (so as to avoid clipping). This compression parameter is specified in dB and nothing stops you from increasing it all the way up to 99dB but that would be utterly ridiculous. It MUST be noted that compression can improve intelligibility but also results in more distorted, less natural sounding speech. So it should be used with caution. I have tried up to 18dB and it was still recognizably me and easy to understand, and much “louder”.

USB sound card audio

QMX has an internal 48ksps 24-bit USB sound card. The SSB transmitter subsystem can take its audio source as the USB sound card instead of the microphone. This is perfect for handling Digital modes such as VARA or PSK31 which are not FSK modes, and therefore cannot be handled by the QMX Digi mode.

When the USB sound card is selected as the source, USB audio samples at 48ksps are converted to a 12ksps audio sample stream for the SSB transmitter. VOX can be used to enable SSB transmission as soon as audio is detected. Or the transmission can be enabled by CAT commands transmitted over the built-in USB Virtual COM Serial port.

Controlled Envelope SSB

Controlled Envelope SSB is a way of removing the amplitude envelope overshoots which occur in all SSB transmitters. CESSB was introduced by David Hershberger W9GER in a 2014 QEX article: http://www.arrl.org/files/file/QEX_Next_Issue/2014/Nov-Dec_2014/Hershberger_QEX_11_14.pdf. This article explains everything. It is vital to understand that SSB amplitude envelope is NOT a problem specific to Digital (SDR) SSB transmitters. It occurs on ANY SSB transmitter, whether an old filter-method SSB tranceiver, phasing analog transceiver, Weaver, ANYTHING.

The effect of having CESSB switched ON, on speech, is typically to increase the effective power of the transmitter by around 4 to 5 dB. So a 5W transmitter will sound to the remote station as though it is running 12W (for example). For single tone (CW), two-tone transmissions, FSK Digi modes, and some others, CESSB does not provide any improvement. It is primarily a technique for improving average to peak power on Speech transmissions.

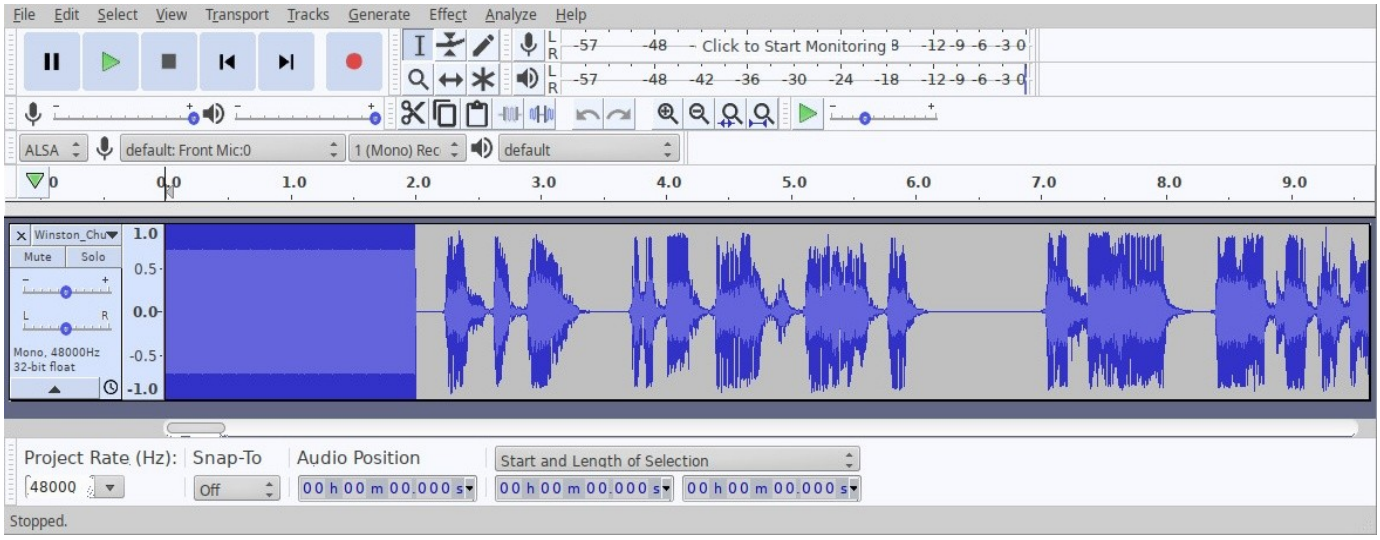
If CESSB is not switched on, in QMX (and in other transceivers), Automatic Level Control (ALC) will act to reduce the gain of the transmitter, to prevent the overshooting peaks of the amplitude envelope from driving the PA into highly non-linear clipping. It is reported that the very best, fast look-ahead ALC systems can produce an improvement of around 2dB. So CESSB is still a clear winner compared to excellent ALC.

In QMX, when CESSB is switched off, a simple ALC action is implemented which prevents peaks from causing RF clipping and temporarily reduces gain, which is held at the lower value and gradually recovers; in very much the same way as the Microphone AGC described above.

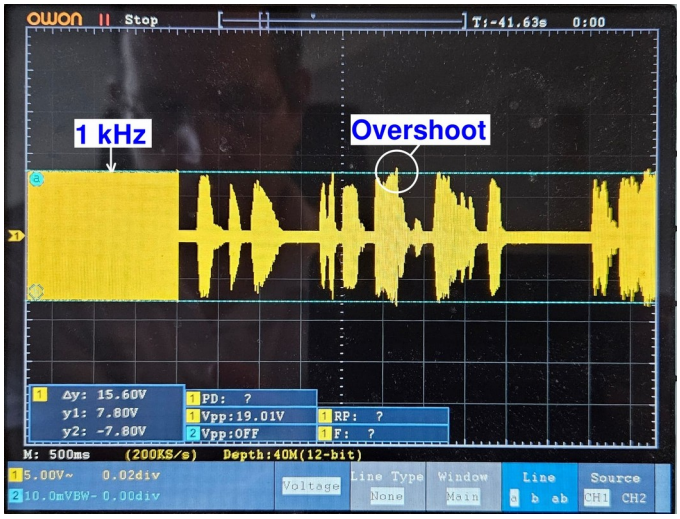
CESSB is a very big deal. It is currently typically only implemented on very high end radios such as Elecraft K4, Apache Anan SDRs and Flex SDR software. Most big brand black box radios do not even support CESSB. To have CESSB implemented on such a low cost, small, QRP radio is unheard of.

To demonstrate the need for CESSB, I used a 10 minute audio clip from the internet (a Winston Churchill speech) loaded into the PC program Audacity.

I then amplified the entire sample such that its peaks were hitting maximum amplitude (notionally +/- 1.0). Audacity contains a tone generator which was used to insert 2 seconds of pure 1 kHz sinewave at full amplitude at the start of the audio clip.



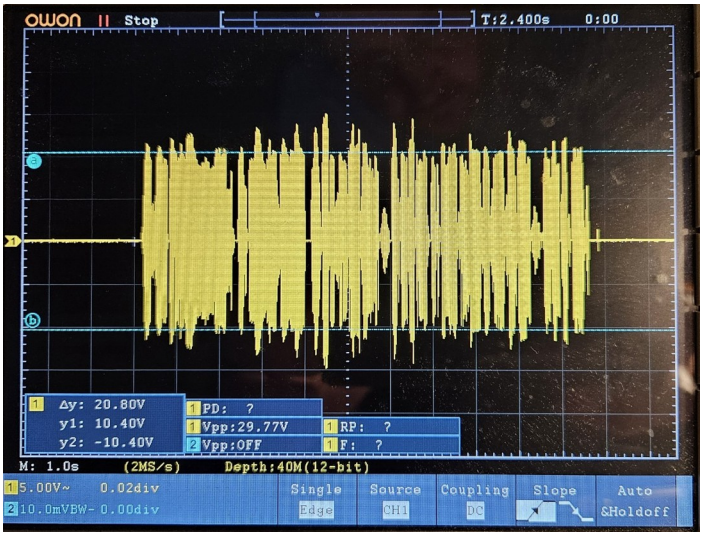
When this clip is applied to the transmitter, with an artificial gain factor applied of about 0.5 such such that there is plenty of space to 4 or 5W PEP; the 1 kHz tone appears and is just a pure, full amplitude carrier, not generating any overshoot). This is shown at the left of this oscilloscope trace.

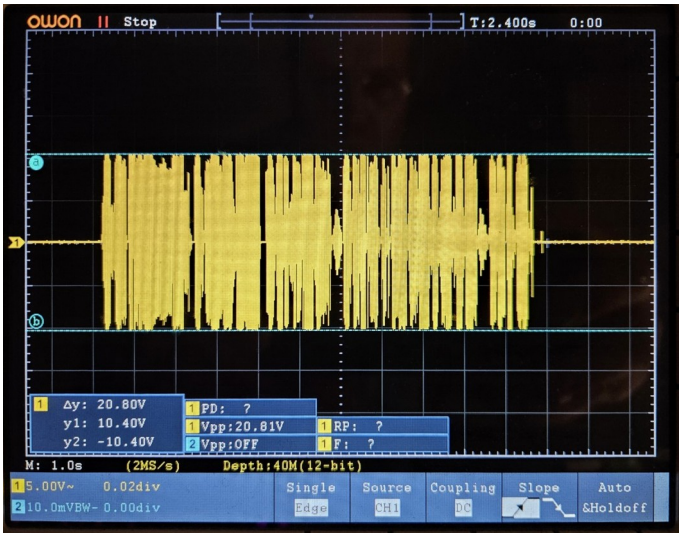


The oscilloscope cursors (horizontal blue lines, here) can then be set to what would be the PEP level (if no 0.5 gain factor had been artificially applied).

Notice that very soon, an overshooting peak is seen. This must not be allowed to limit the RF amplifier or that would create splatter (adjacent channel interference) so an ALC (Automatic Level Control) system will act to reduce the gain to prevent that. The gain reduction action also reduces the average power of the whole transmission.

One may also experiment with the audio file used for the QEX CESSB article, to reproduce the effects demonstrated in the article. The screenshot below left shows the overshoots inflicted by the SSM modulation process. The screenshot below right, is with CESSB processing applied. All overshoots are eliminated so the ALC will not need to reduce gain to compensate.





Conclusion

The QMX SSB project really tested my patience and endurance. But with dogged determination and not least, the knowledge that my reputation depended on it, I managed to push through to the end. I learned many new techniques and made discoveries along the way that will steer my future transceiver designs. The performance was far better than I had originally hoped for even in my most optimistic moments.

Fundamentally what excites me most is that it changes the way I think about SSB transmitters. Conventionally designers fight with transistor curves, expending untold effort in trying to make curves into straight lines, achieving highly linear amplification. With polar modulation (EER), just as Kahn said 73 years ago, we can obtain excellent performance (signal purity measured in terms of low intermodulation spurious products) without worrying about linearity. The linearity problem is moved to audio frequencies where it is much more easily achieved using more elaborate feedback amplifiers. The use of the ‘5351 frequency synthesizer to effect phase modulation, demonstrated by Guido PE1NNZ, neatly makes the phase modulation part of the implementation very easy and precise to achieve also.

I really hope that in coming years we will see more of this SSB generation technique in amateur radio equipment.