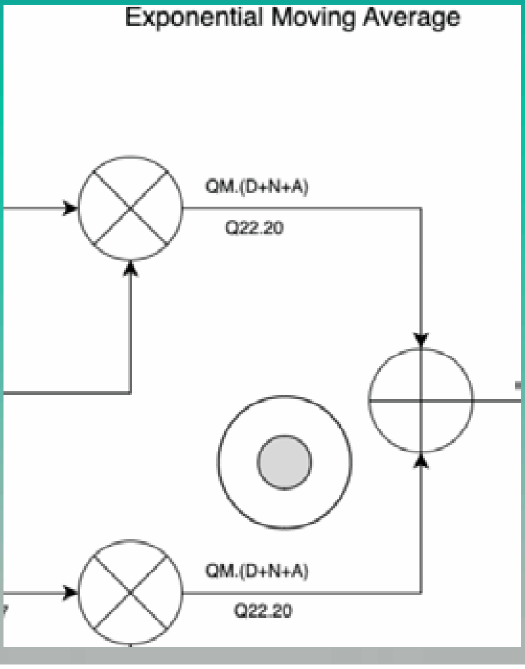
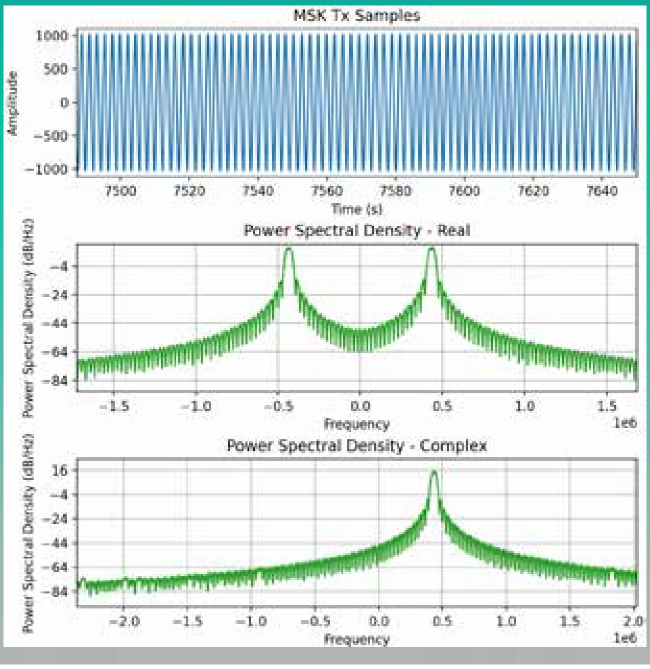
Open Research Institute Inner Circle Newsletter January 2025

Exponential moving what? See page 2!



Opulent Voice moves from real to complex modulation. See page 6 to find out why.



The Who What When Where Why

Open Research Institute is a non-profit dedicated to open source digital radio work. We do both technical and regulatory work. Our designs are intended for both space and terrestrial deployment. We're all volunteer.
You can get involved by visiting https://openresearch.institute/getting-started

Membership is free. All work is published to the general public at no cost. Our work can be reviewed and designs downloaded at https://github.com/OpenResearchInstitute

We equally value ethical behavior and over-the-air demonstrations of innovative and relevant open source solutions. We offer remotely accessible lab benches for microwave band radio hardware and software development. We host meetups and events at least once a week. Members come from around the world.

OPEN SOURCE HARDWARE AND SOFTWARE NEWSLETTER SIGNUP
FREE AND 100% VOLUNTEER DRIVEN

Want more Inner Circle Newsletters?
Use the QR code at left or go to
http://eepurl.com/h_hYzL
and sign up.

What's all this Fixed-Point Math Stuff, Anyhow?

With apologies to Bob Pease

Person 1: Hey, whatcha doing? Looks like something cool.

Person 2: Working on a Simulink model for an MSK model.

Person 1: Oh, that's fun, how about I code up the modem in VHDL this weekend?

Person 2: Sounds great!

Famous last words, amirite? It has been many months - *ahem* 10 - since that pseudo conversation occurred. In that time there have been missteps, mistakes, and misery. We have come a long way, with internal digital and external analog loopback now working consistently, although unit to unit transmission isn't quite there yet.

This is *one* way to do hardware development. Write some HDL code, simulate, and iterate. Once the simulation looks good, put it in an FPGA, test it, and iterate. You'll get there eventually, but there will be some hair pulling and teeth gnashing along the way. But, it is worth the effort when you finally see it working, regardless (and because) of all the dumb mistakes made along the way.

There are many ways to approach a design problem, some better than others, and as with all things engineering, it's a trade-off based on the overall design context. At opposing ends of the spectrum we have empirical and theoretical approaches. Empirical: build based on experience, try it, fix it, rinse and repeat. And, theoretical: build based on theory, try it, fix it, etc. Ultimately we must meet in the middle (and there is never getting away from the testing and iterating).

Starting from first principles always serves us well. One of the niggling points in the Minimum-Shift Keying (MSK) development has been selection of bit-widths for the signal processing chains.

Person 1 (yeah, that's me): I'll code it up this weekend. Let's see, for the modulator we need data in, that's 1-bit. The data gets encoded, still 1-bit. The data modulates a sine wave, hmmm, how many bits should that be? Well the DAC is 12-bits, so we should use a 2's complement 12-bit number. That seems right.

Person 1: Now for the demodulator. The ADC outputs 12-bits, so a 2's complement 12-bit number. That gets multiplied by a sine wave, let's use a 2's complement 12-bit number since we did that in the modulator. The multiply output is 24-bits, ok. Now we integrate that 24-bit number over a bit-period, hmmm, how many bits after the integration? No worries, let's just make it 32-bits and keep on. But that is a lot of bits, *let's just scale the 32-bits down to 16-bits and keep going. Ah, the empirical approach, I'm getting so much done!*

Person 1: Why isn't this working? Oh, the 16-bit number is overflowing. I wish I could just make it 32-bits again, but that's too many bits for the multiplier.

There is a better way! We can analyze the design from some starting point, such as, the number of input bits. From there we operate on those bits, adding, subtracting, multiplying, shifting, etc. And we know how each of those operations affects the bit widths, which allows us to choose appropriate bit-widths through the signal chain. Before we get there we need to take a quick look at fixed point math and related notation.

In our everyday base 10 math we have, essentially, infinite precision. One example is that irrational numbers can be expressed as fractions, and we can operate on those fractions and the infinite precision is maintained as long as we can keep the fractional representation. Often, though, we will need to use an approximation of the fraction to get an actual answer to a problem. One example is $1/3 = 0.33333...$ and we have to choose how many digits of 3 we need for the particular problem at hand. And, we know there is an error term when we use such an estimation ($1/3 = 0.333 + \epsilon$).

Base 2 math is largely the same, but hardware can't use fractional representations, meaning that an irrational number will always be a finite representation. Also, we may be constrained in how many bits we can use to represent numbers. We need a way to notate these numbers.

Texas Instruments created Q-notation as a way to specify fixed-point numbers. The notation *Qm.n* is used to represent a signed 2's complement number where m is the number of whole bits and n is the number of fractional bits. TI specifies m to not include the sign-bit, while ARM specifies m to include the sign bit. The table below shows examples of Q-notation using both TI and ARM variants.

Table 1. Q-Notation Examples

Total Bits	ARM Q Format	TI Q Format	Range	Resolution
8	Q3.5	Q2.5	-4.0 to +3.96875	2^{-5}
16	Q4.12	Q3.12	-8.0 to +7.9997558594	2^{-12}
24	Q8.16	Q7.16	-128.0 to +127.9999847412	2^{-16}
32	Q8.24	Q7.24	-128.0 to +127.9999999404	2^{-24}
8	UQ8.3	UQ8.3	0.0 to +255.875	2^{-3}
10	Q1.10	Q10	-1.0 to +0.9990234375	2^{-10}

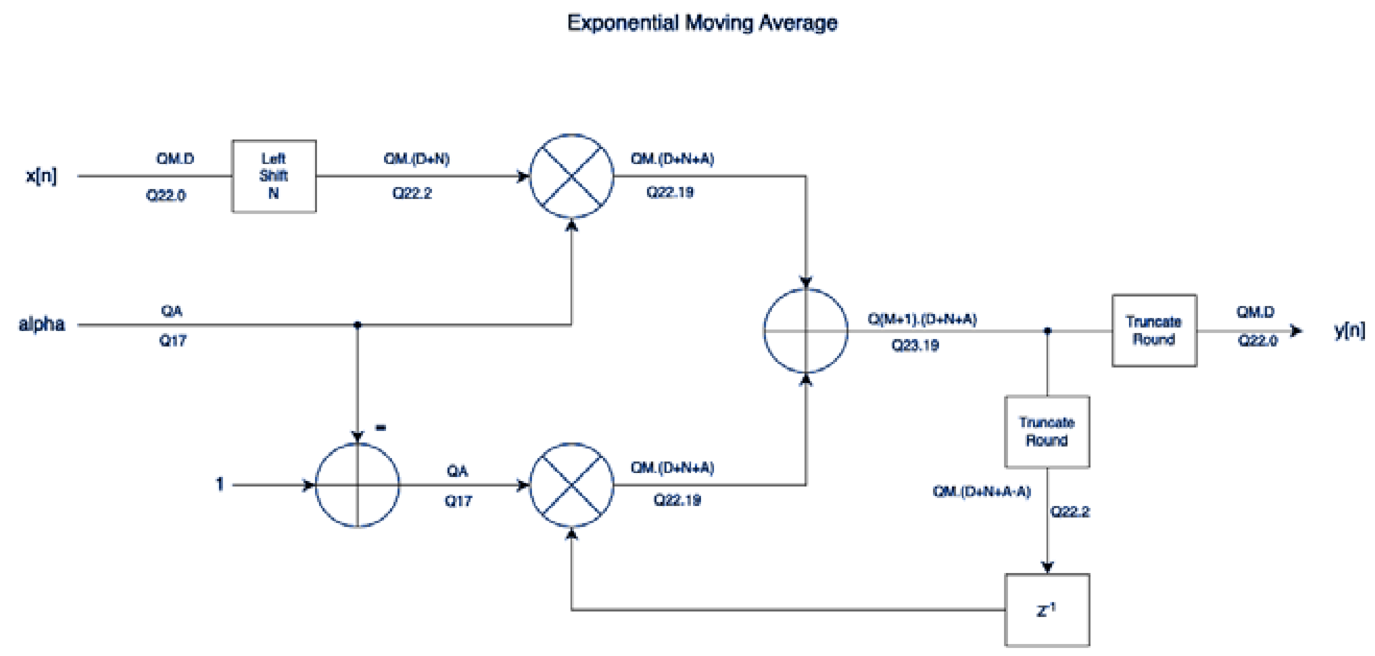
The table below shows how bit widths are affected by various operations.

Table 2. Math operators affect on bit-widths

Operation	Input Numbers	Output
+/-	Qm.n +/- Qx.y	Qj.k where j = max(m,x)+1 and k = max(n,y)
*	Qm.n * Qx.y	Q(m+x).(n+y)
>>	Qm.n >> k	Qm.(n-k)
<<	Qm.n << j	Qm(n+j)

The diagram below shows an exponential moving average circuit with signal bit-widths notated as Qm.d. You can see the bit widths adjusted as we shift, multiply, add, etc. Since the output is an average of the input, it should have the same representation as the input (Qm.d). The lower representation (i.e. Q22.0 for the input) is an actual bit-width selection based on targeting the Zynq7010 and its 25x18 hardware multipliers. Some particular notes:

1. The Q22.0 input is specified by the surrounding system. It is left-shifted by 2-bits (Q22.2) to fully utilize the 25-bit multiplier input thus increasing the resolution.
2. The alpha input is specifically chosen to be Q17 to fully utilize the 18-bit multiplier input.
3. The bottom multiplier is in a feedback path, its output must match the upper multiplier output so that the binary points are aligned into the adder. To this end the adder output is truncated and rounded by 18-bits.



NOTES:
1. Lower Qm.n values are for a a 25x18 multiplier (Zynq7010)
2. Using Texas Instruments Q-notation: sign bit is not included in the m value. Total bits is 1+m+n.

Figure 1. Exponential Moving Average Block Diagram

The adder output is truncated and rounded by 20-bits for the final output.

In summary Q-notation is a useful tool for understanding and specifying system bit-widths throughout the processing chain. It is especially useful to add Q-notation to the block diagram to help visualize the bit-widths. With this approach the optimal bit-widths should become apparent when taking into account system requirements. Doing this system analysis step before writing any code will save time and effort by reducing errors. The other benefit is that device resources are not over utilized, which may make the difference between fitting in an FPGA or not.

Additional reading and resources are available at these URLs:

- Read more about Q-notation at Q (number format) - Wikipedia
- Try out the Q-notation calculator at Q-format Converter & Calculator
- A Python package for Fixed-point math is at fixedpoint 1.0.1 Package Documentation including explanations of Q-notation and how bit-widths are affected by arithmetic.

As the saying goes, *mind your Ps and Qs.*

Matthew Wishek, NB0X

Real and Compl

The magic of radio is rooted in mathematics. Some of that math can be complicated or scary looking. We are going to break things down bit by bit, so that we can better understand what it means when we say that we are going to transmit a complex baseband signal.

Everything that we are going to talk about today is based on a single carrier real signal, even when we get to complex transmission. A single carrier real signal is where we take our data, a single-dimensional value that we want to communicate, and we multiply it by a carrier wave (a cosine wave) at a carrier frequency (fc). Let's call the value we want to communciate "alpha".

Because we are dealing with digital signals, the value that we are transmitting is held for a period of time, called T. The next period of time we send another constant value. And, so on. We are sending discrete values for a period of time T, one after another until we are all done sending data, and not continuous values over time.

Let's say we are sending four different amplitudes to represent four different values. During each time period T we select one of these four amplitude values. We hold that value for the entire time period. These values can be thought of as single dimensional values. One value uniquely identifies the value we want to send. In this case, amplitude.

Sending one of four values at a time means we are sending two bits of data at a time.

alpha	bits
0	00
1	01
2	10
3	11

In order to send our value out over the air from transmitter to receiver, we multiply our alpha by our carrier frequency. The result is alpha*cos(2*pi*fc*t). Cosine is a function of time t. The 2*pi term converts radians per second to cycles per second, which is something that most of us find easier to deal with. When we multiply in the time domain, we cause a different mathematical thing to happen in the frequency domain. Multiplication is called convolution in the frequency domain. This mathematical process creates images, or copies, of our baseband signal in the frequency domain. One image will be located at fc and the other will be located at -fc.

Our real signal has a special characteristic. It's symmetric. At the receiver, we multiply what we receive by that same cosine wave, cos(2*pi*fc*t). We multiply in the time domain, and we convolve in the frequency domain. This results in images at 2*fc, -2*fc, and most useful to us, we get two images at 0 Hz. We use a low-pass filter to get rid of the unwanted images at 2*fc and -2*fc, and by integrating over the time period T, we get a scaled version of the original value (alpha) that was sent. Amazing! We reversed the process and we got our original sent value.

So what's all this complex signal stuff all about? Why mess with success? We have our single carrier signal and our four values. What more could we want?

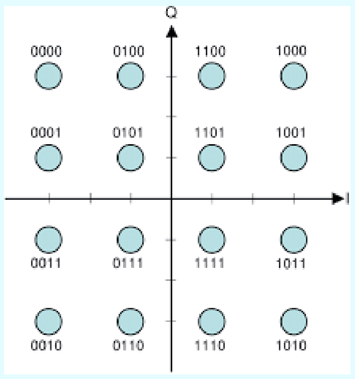
Well, we want to be able to send more than a shave and a haircut number of bits!

If we want to send more bits in the same time period (and who doesn't?) then we must use a bigger alphabet. Let's double our throughput. We now pick from sixteen different amplitudes, sending the value we picked out for a period of time T as a single-carrier real signal. Now, each alpha value stands for four bits.

lex Signal Basics

We have a minor problem. Sending out sixteen different voltage levels on a single carrier means that we have to be able to differentiate between finer and finer resolution at our receiver. Before, we only had to distinguish between four different levels. Now we have sixteen. This means we better have a really clear channel and a lot of transmit power. But, we don't always have that. It's expensive and a bit unreasonable. There is a better way.

We know we now want to send out (at least) one of sixteen values, not just one of four. If we turn our one-dimensional problem into a two-dimensional problem, and assign a real single carrier signal to, say, the vertical dimension, and then a second real single carrier signal to the horizontal dimension, then we are now enjoying the outer limits of digital signal processing. The vertical handles four levels. The horizontal handles four levels. We still have the same time period T. We just have a two-dimensional coordinate system instead of a one-dimensional coordinate system.



16QAM signal diagram

alpha	bits	alpha	bits
0	0000	8	1000
1	0001	9	1001
2	0010	10	1010
3	0011	11	1011
4	0100	12	1100
5	0101	13	1101
6	0110	14	1110
7	0111	15	1111

But how can we send two real signals over the air, at the same time? We can't just add them together, can we? They will step on each other and we'll get a noisy mess at the receiver. Math saves us! We can actually add these two signals together, send them as a sum, and then extract each dimension back out. But, only if we prepare them properly. And here is how that is done.

Look at the two-dimensional diagram of 16QAM. The vertical axis is labeled Q, and the horizontal axis is labeled I. When we want to indicate the vertical dimension of our value (pick any one of them), then we take that vertical dimension (say, -1 for 1111) and we multiply it by sin(2*pi*fc*t). We now have our Q signal. Now we need the horizontal location of 1111. That would be +1 on the I axis. We multiply this value, giving the horizontal dimension, by cos(2*pi*fc*t). We now have our I signal. Q axis value was multiplied by sine. I axis value was multiplied by cosine. These signals are played for the duration of the sample period. Both of them happen at the same time to give a coordinate pair for a particular alpha.

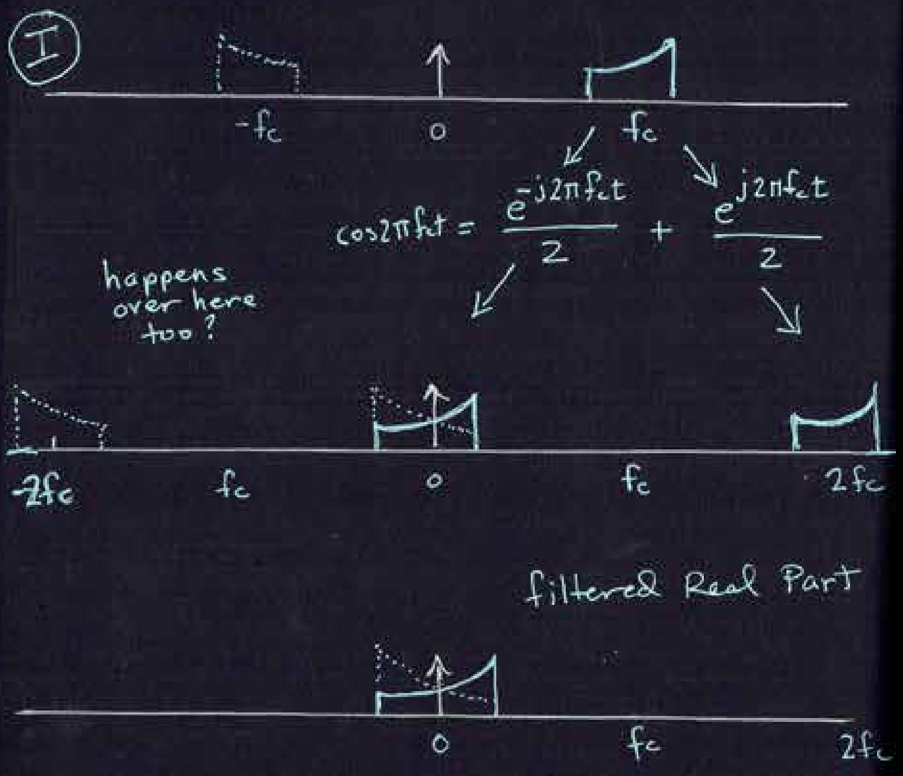
We add the I and Q signals together and transmit them. We are sending (I axis value) * cos(2*pi*fc*t) + (Q axis value) * sin(2*pi*fc*t).

At the receiver, we take what we get and we split the signal. We now have two copies of what we received. We multiply one copy by cos(2*pi*fc*t). We multiply the other by sin(2*pi*fc*t). We integrate over our time period T. This is important because it lets us take advantage of several trig identities.

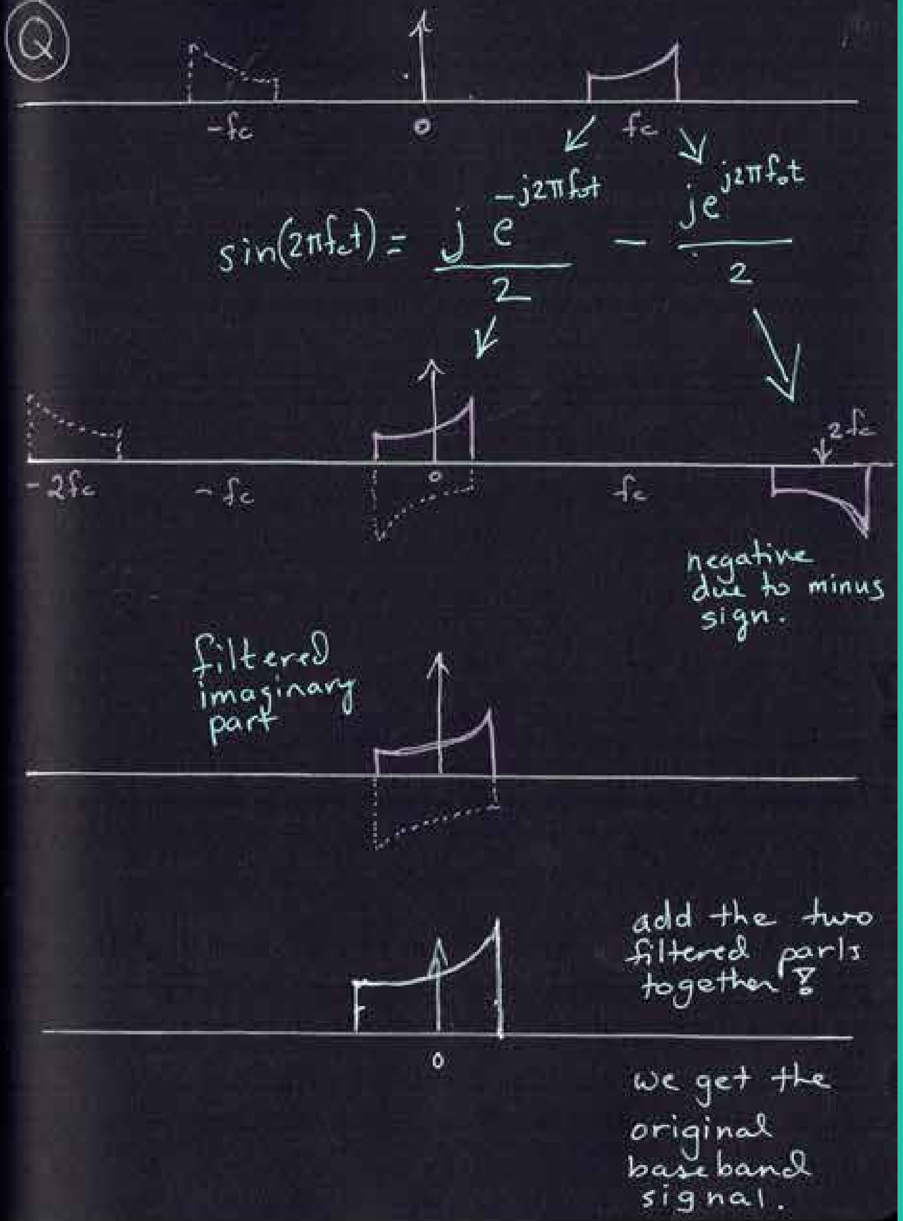
First, let's multiply and distribute our cos(2*pi*tfc*t) across the summed signals we received. We multiply:

[(I axis value) * cos(2*pi*fc*t) + (Q axis value) * sin(2*pi*fc*t)] * cos(2*pi*fc*t)

And rewrite it to distribute our cos(2*pi*fc*t).



$$\cos 2\pi f_c t = \frac{e^{-j2\pi f_c t}}{2} + \frac{e^{j2\pi f_c t}}{2}$$



$$\sin(2\pi f_c t) = \frac{j\, e^{-j2\pi f_c t}}{2} - \frac{j e^{j2\pi f_c t}}{2}$$

add the two filtered parts together ☟

we get the original baseband signal.

(I axis value) * cos^2(2*pi*fc*t) + (Q axis value) * sin(2*pi*fc*t)*cos(2*pi*fc*t)

Aha! We can convert that cos^2() term to something we can use. Use the half angle identity, square each side, and double all the angle measurements (easy, right?). After this cleverness, this is what we have.

cos^2(2*pi*fc*t) = 1/2*[1 + cos(2*pi*2fc*t)]

So now we have

(I axis value) * 1/2*[1 + cos(2*pi*2fc*t)]

See that 2fc term in there? Check out the notebook drawing for our signal in the frequency domain. It's at 2fc. Q signal is on the right-hand half of the drawing.

Let's rearrange things.

(I axis value/2) + [(I axis value/2) * cos(2*pi*2fc*t)]

Remember we are integrating over time at the receiver. We have one of the two terms rewritten in a useful way. What happens when we integrate a cosine signal from 0 to T? That value happens to be zero! This leaves just the integration of (I axis value/2)!

The result at the receiver for the multiplication and integration of the first copy of the received signal is (I axis value)*(T/2). We know T, we know what the number 2 is, so we know the I axis dimension value.

But wait! We forgot something. We only did the first part.

Remember we had

(I axis value) * cos^2(2*pi*fc*t) + (Q axis value) * sin(2*pi*fc*t)*cos(2*pi*fc*t)

We recovered I axis value from the term before the plus sign. But what about the term after the plus sign?

(Q axis value) * sin(2*pi*fc*t)*cos(2*pi*fc*t)

Uh oh we didn't get away from summing the I and Q together after all...

Trig saves us here too. When we integrate sin(2*pi*fc*t)*cos(2*pi*fc*t) from 0 to period T, it happens to be zero. The entire Q axis value term drops out. Does the same technique work for the copy of the received signal that we multiply by sin(2*pi*fc*t)?

You bet it does! First, let's multiply and distribute our sin(2*pi*tfc*t) across the second copy of the summed I and Q signals we received. We multiply:

[(I axis value) * cos(2*pi*fc*t) + (Q axis value) * sin(2*pi*fc*t)] * sin(2*pi*fc*t)

And rewrite it to distribute our cos(2*pi*fc*t).

(I axis value) * cos(2*pi*fc*t) * sin(2*pi*fc*t) + (Q axis value) * sin(2*pi*fc*t)*sin(2*pi*fc*t)

Now that we know that integrating cos(2*pi*fc*t) sin(2*pi*fc*t) from 0 to T is zero, we can drop out the I axis value term. That's good because we already have it from multiplying our received summed signal by cos(2*pi*fc*t) and doing trigonomtry tricks.

We are left with

(Q axis value) * sin(2*pi*fc*t)*sin(2*pi*fc*t)

And we rewrite it

(Q axis value) * sin^2(2*pi*fc*t)

And use the half angle trig identity, square each side, and then double all angle measurements.

We can replace sin^2(2*pi*fc*t) with

1/2*[1 - cos(2*pi*2fc*t)]

which gives us

(Q axis value) * 1/2*[1 - cos(2*pi*2fc*t)]

And we rewrite it as

(Q axis value/2) - [(Q axis value/2)*cos(2*pi*2fc*t)]

Hey, guess who goes to zero again? That's right, cosine integrated from 0 to T is zero. We are left with a constant term that integrates out to (Q axis value) * (T/2)

So when we multiply the summed signal that we received by cosine, we get I axis value. When we multiply the summed signal that we received by sine, we get Q axis value.

I and Q give us the coordinates on the 16 QAM chart. As long as we are in sync with our transmitter (a whole other story) and as long as our map of which point stands for which label (read your documentation!) is the same as at the transmitter, then we have successfully received what was sent using a technique called quadrature mixing.

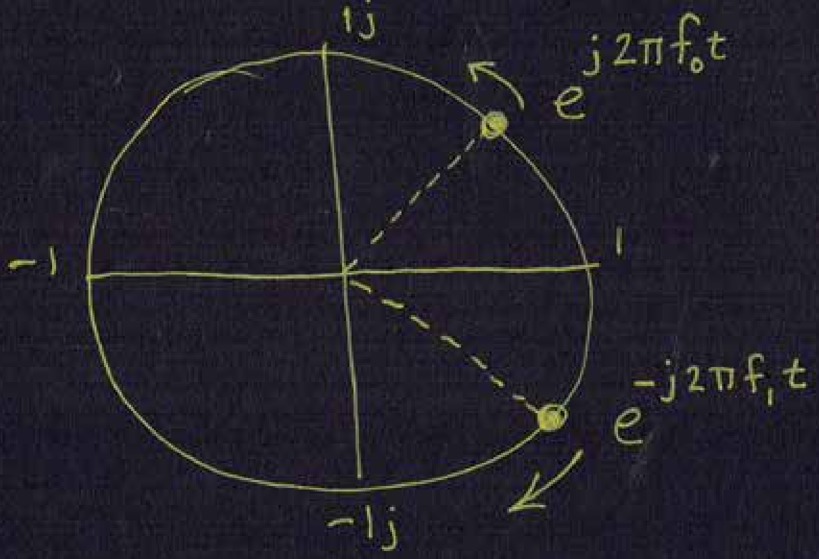
Moving from a single carrier real signal to a "complex" signal, where two real signals are sent at the same time using math to separate them at the receiver, gives us advantages with respect to sending more bits without having to send more levels. Our two signals are each handling four levels, but using the results in a two-dimensional grid gives us more bits per unit time without having to change our performance expectations. Sending sixteen different levels is harder than sending four. So, we send four twice and use some mathematical cleverness.

However, doing this complex modulation scheme gives us yet another advantage. Because of the math we just did, we eliminate an entire image when compared to a single carrier real signal. We have a less difficult time with filters because we no longer create a second image. Below (next page) are some diagrams of how this happens.

A third advantage of I and Q modulation is that it doesn't just do things like 16QAM. Using an I and a Q, and a fast enough sample period T, means you can send any type of modulation or waveform. Now that's some power!

This technique does require some signal processing at the receiver. But, this type of signal handling is at the heart of every software defined radio. And, now you know how it's done, and the reasons why Opulent Voice is now using complex modulation in the PLUTO SDR implementation.

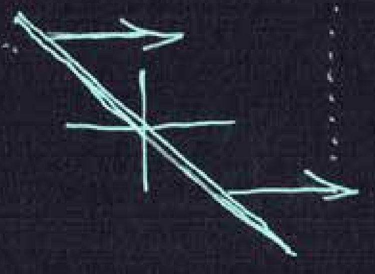
-Michelle Thompson W5NYV

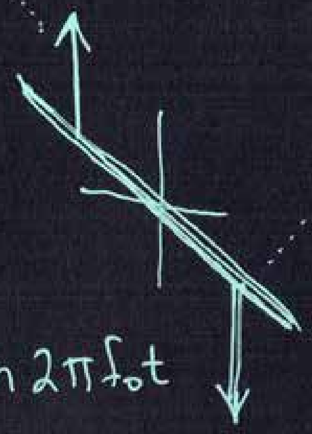


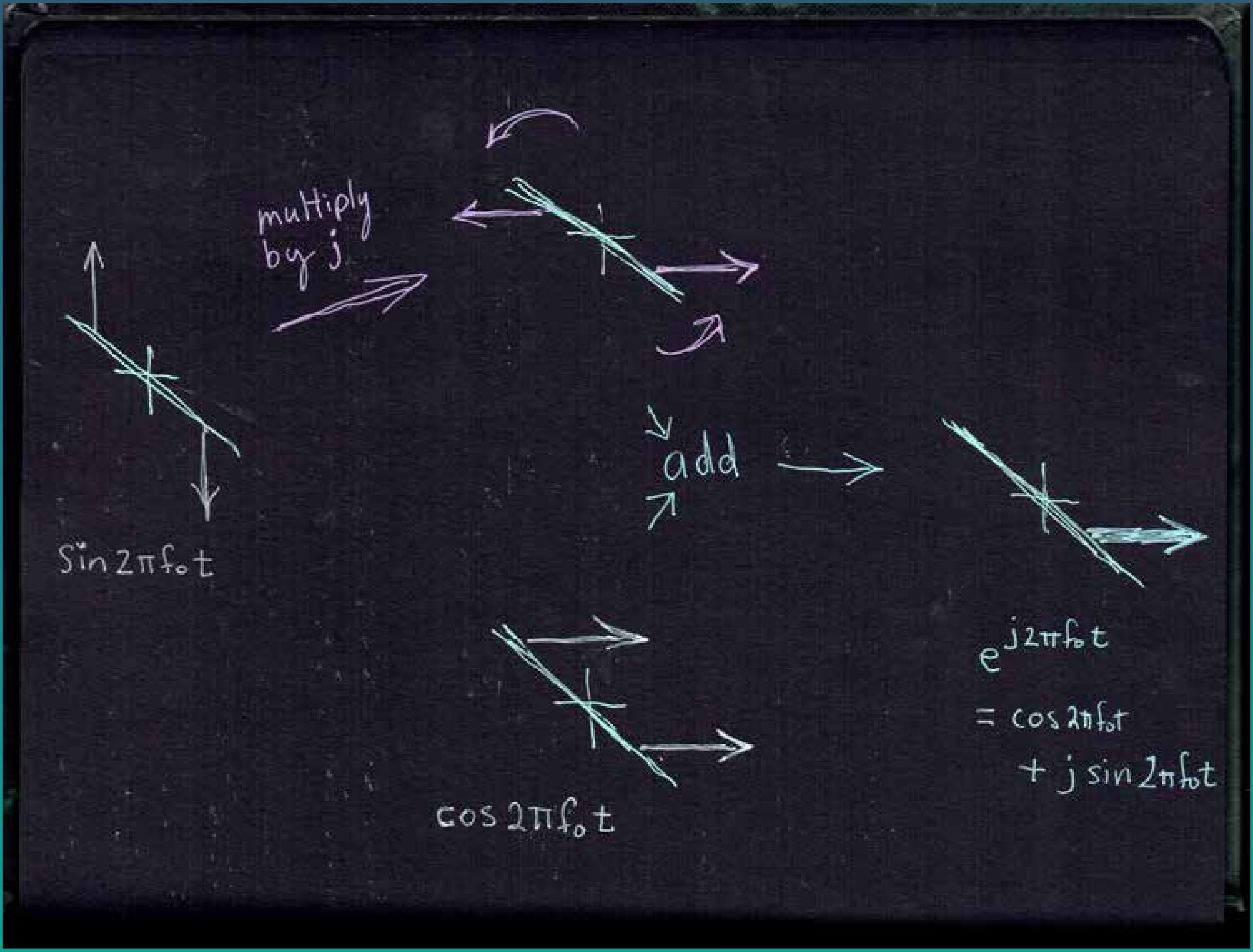
sum these up and you get a real signal because the imaginary parts cancel out.

$$\cos(2\pi f_0 t) = \frac{e^{j2\pi f_0 t}}{2} + \frac{e^{-j2\pi f_0 t}}{2}$$

$$\sin(2\pi f_0 t) = \frac{j\,e^{-j2\pi f_0 t}}{2} - \frac{j\,e^{j2\pi f_0 t}}{2}$$






multiply by j
add
$\sin 2\pi f_0 t$
$\cos 2\pi f_0 t$
$e^{j2\pi f_0 t}$
$= \cos 2\pi f_0 t$
$+ j \sin 2\pi f_0 t$

Adding a Preamble to Opulent Voice

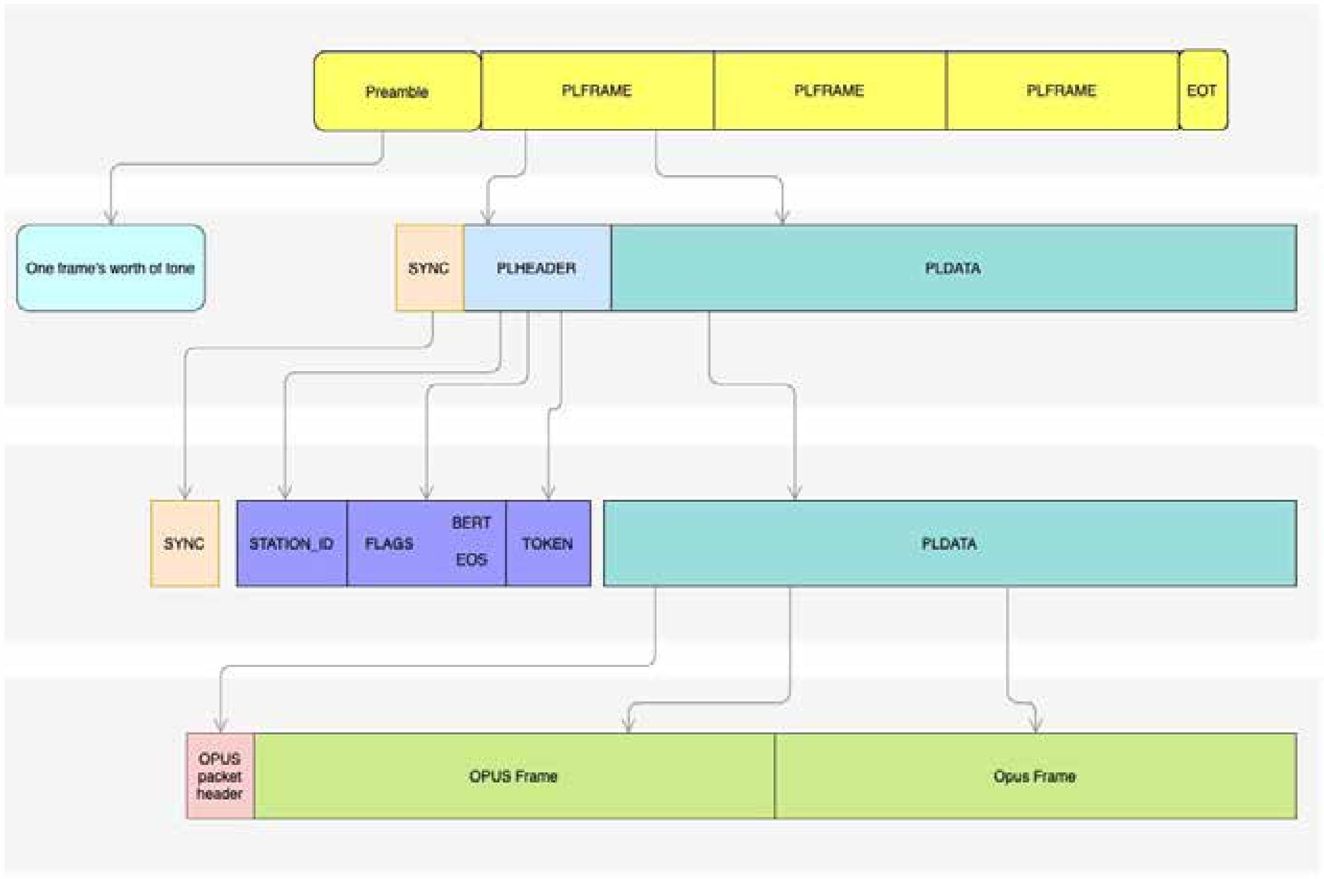
Looking at the Opulent Voice protocol overview diagram below, we can see that each transmission begins with a preamble. This section of the transmission contains no data, but is extremely helpful in receiving our digital signal.

The preamble is like a lighthouse for the receiver, revealing a shoreline through the fog and darkness of interference and noise. While we may not need the entire 40 milliseconds of preamble signal to acquire phase and frequency, so that we are "on board" for the rest of the transmission, keeping the preamble at the length of a frame simplifies the protocol.

There is a similar end of transmission (EOT) frame, so that the receiver knows for sure that the transmitted signal has ended, and has not simply been lost. This will reduce the uncertainty at the receiver, and allow it to return to searching for new signals faster and more efficiently.

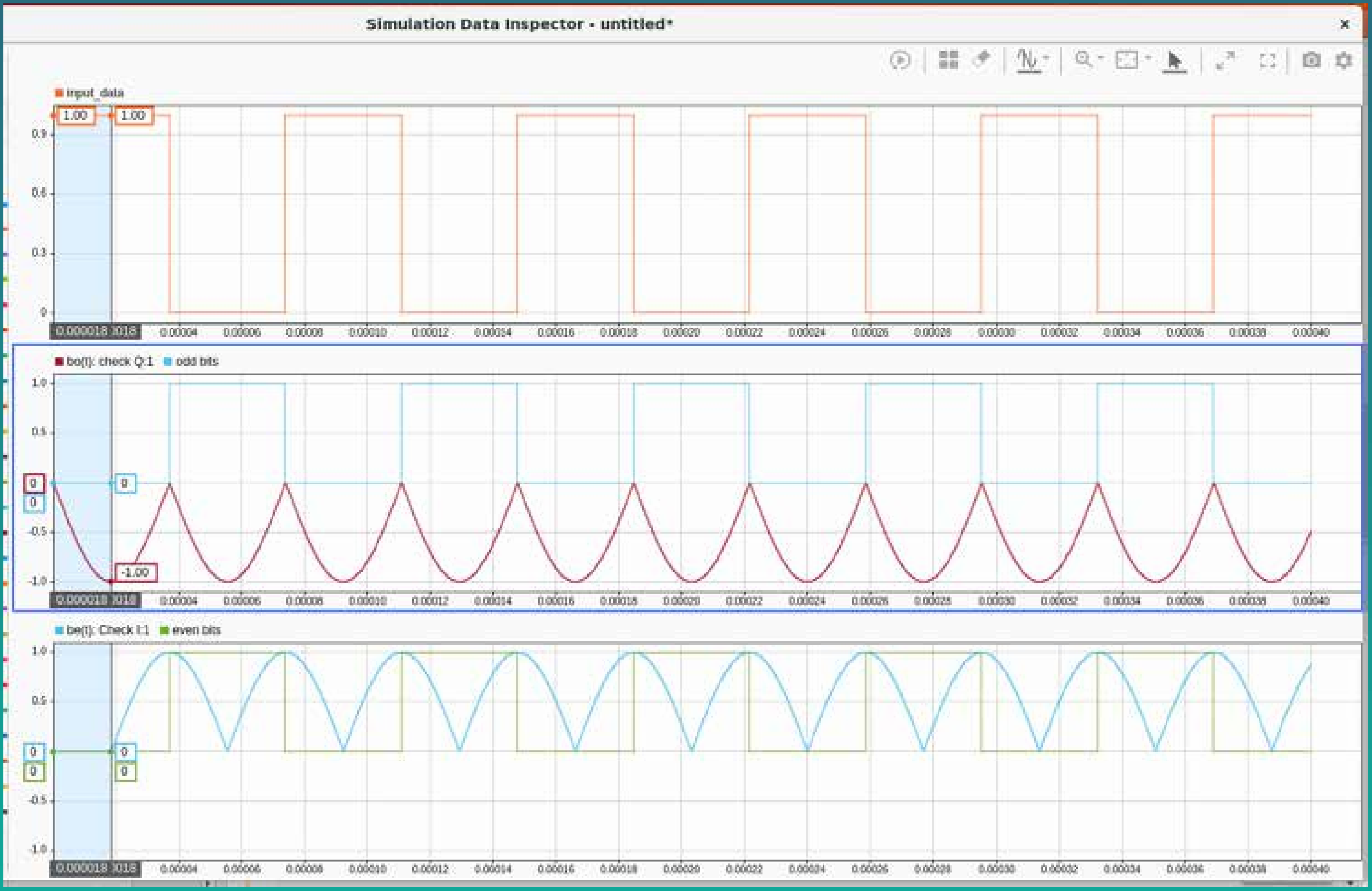
For minimum shift keying, the modulation of Opulent Voice, a recommended preamble data stream in binary is 1100 repeating. In other words, we'd get a frame's worth of 11001100110011001100... at 54.2 kilobits per second for 40 milliseconds.

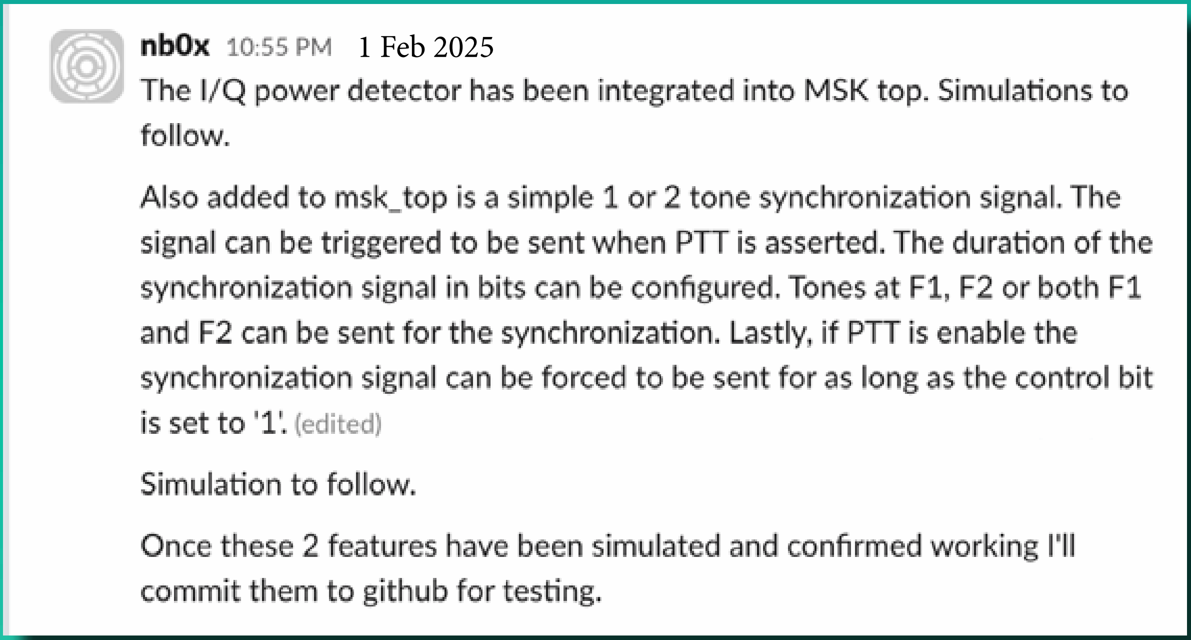
After the preamble is sent, data frames are sent. Note that there is a synchronization segment at the beginning of each frame. This keeps the receiver from drifting and improves reliability.



Constructing the Preamble in Simulink and in HDL

Below is the Simulink model output viewer showing the 1100 repeating pattern mathematical construction, followed by the planned update for the hardware description language (HDL) code updates. The target for HDL firmware is the PLUTO SDR. -Opulent Voice Team



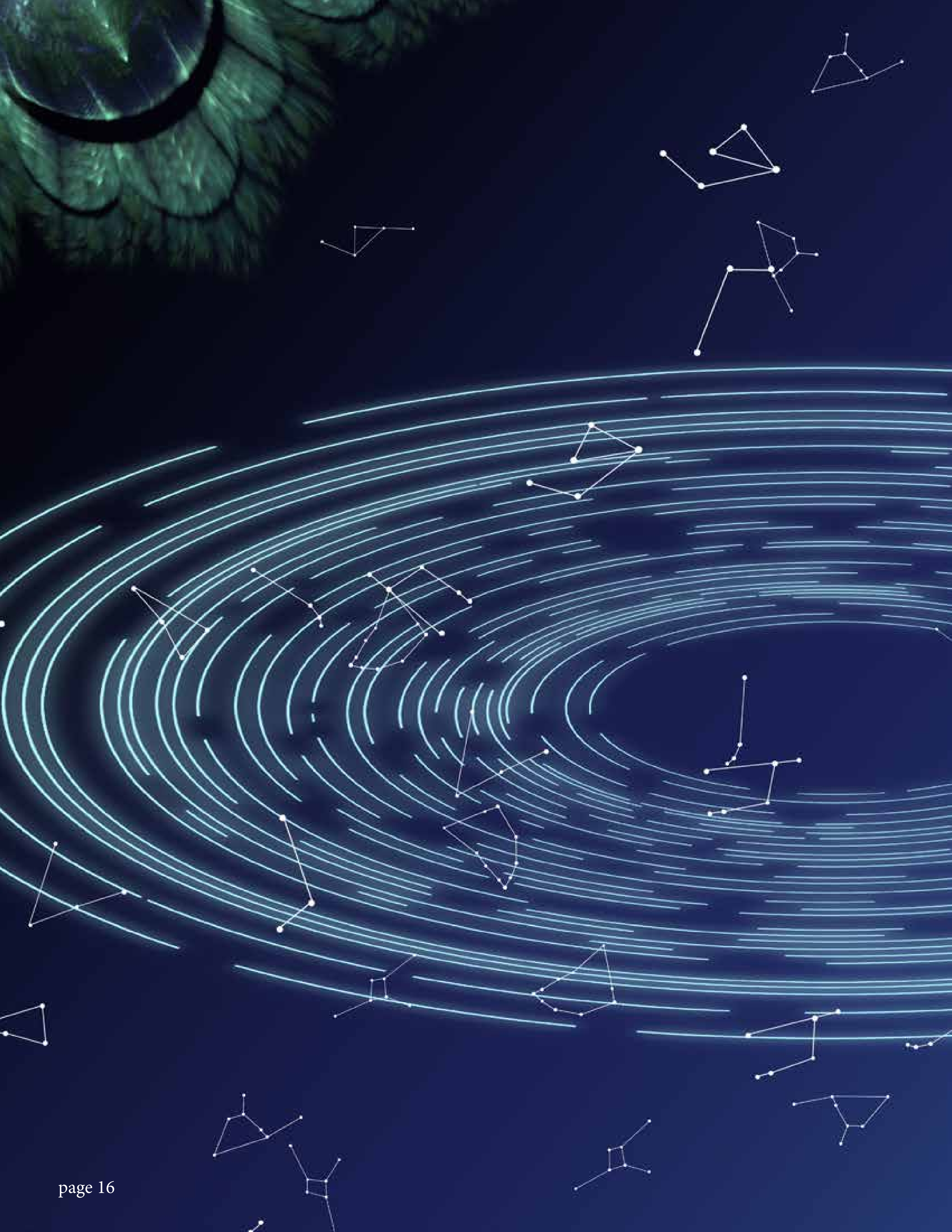

nb0x 10:55 PM 1 Feb 2025

The I/Q power detector has been integrated into MSK top. Simulations to follow.

Also added to msk_top is a simple 1 or 2 tone synchronization signal. The signal can be triggered to be sent when PTT is asserted. The duration of the synchronization signal in bits can be configured. Tones at F1, F2 or both F1 and F2 can be sent for the synchronization. Lastly, if PTT is enable the synchronization signal can be forced to be sent for as long as the control bit is set to '1'. (edited)

Simulation to follow.

Once these 2 features have been simulated and confirmed working I'll commit them to github for testing.



“Take This Job”

Interested in Open Source software and hardware? Not sure how to get started? Here’s some places to begin at Open Research Institute. If you would like to take on one of these tasks, please write hello@openresearch.institute and let us know which one. We will onboard you onto the team and get you started.

Opulent Voice:

- Add a carrier sync lock detector in VHDL. After the receiver has successfully synchronized to the carrier, a signal needs to be presented to the application layer that indicates success. Work output is tested VHDL code.
- Bit Error Rate (BER) waterfall curves for Additive White Gaussian Noise (AWGN) channel.
- Bit Error Rate (BER) waterfall curves for Doppler shift.
- Bit Error Rate (BER) waterfall curves for other channels and impairments.
- Review Proportional-Integral Gain design document and provide feedback for improvement.
- Generate and write a pull request to include a Numerically Controlled Oscillator (NCO) design document for the repository located at https://github.com/OpenResearchInstitute/nco.
- Generate and write a pull request to include a Pseudo Random Binary Sequence (PRBS) design document for the repository located at https://github.com/OpenResearchInstitute/prbs.
- Generate and write a pull request to include a Minimum Shift Keying (MSK) Demodulator design document for the repository located at https://github.com/OpenResearchInstitute/msk_demodulator
- Generate and write a pull request to include a Minimum Shift Keying (MSK) Modulator design document for the repository located at https://github.com/OpenResearchInstitute/msk_modulator
- Evaluate loop stability with unscrambled data sequences of zeros or ones.
- Determine and implement Eb/N0/SNR/EVM measurement. Work product is tested VHDL code.
- Review implementation of Tx I/Q outputs to support mirror image cancellation at RF.

Haifuraiya:

- HTML5 radio interface requirements, specifications, and prototype. This is the user interface for the satellite downlink, which is DVB-S2/X and contains all of the uplink Opulent Voice channel data. Using HTML5 allows any device with a browser and enough processor to provide a useful user interface. What should that interface look like? What functions should be prioritized and provided? A paper and/or slide presentation would be the work product of this project.
- Default digital downlink requirements and specifications. This specifies what is transmitted on the downlink when no user data is present. Think of this as a modern test pattern, to help operators set up their stations quickly and efficiently. The data might rotate through all the modulation and coding, transmititng a short loop of known data. This would allow a receiver to calibrate their receiver performance against the modulation and coding signal to noise ratio (SNR) slope. A paper and/or slide presentation would be the work product of this project.

YouTube

https://www.youtube.com/@OpenResearchInstituteInc

RFBitBanger

HF Digital QRP
HF Digital QRP
HF Digital QRP
HF Digital QRP
HF Digital QRP
HF Digital QRP
HF Digital QRP
HF Digital QRP

Get your kit today
while supplies last!

https://www.ebay.com/usr/openresearchinstitute

$175

Shipping
Calculated
Separately

The Inner Circle
Sphere of Activity

January 6, 2025 - All labs re-opened. Happy New Year!

January 13, 2025 - ORI presented to Deep Space Exploration Society about our history and projects line-up.

January 18, 2025 - San Diego Section of IEEE Annual Awards Banquet. ORI volunteers supported this event as a media and program sponsor. ORI was represented by five members.

January 23-26, 2025 - IEEE Annual Meeting for Region 6 and Region 4, ORI was represented by three members.

January 28, 2025 - Co-hosted the IEEE Talk "AI/ML Role in RTL Design Generation" with the Information Theory Society and the Open Source Digital Radio San Diego Section Local Group.

February 18, 2025 - San Diego County Engineering Council Annual Awards Banquet. ORI will be part of the IEEE Table display in the organizational fair held on site before dinner. ORI will be represented by at least one member.

Thank you to all who support our work! We certainly couldn't do it without you.

Anshul Makkar, Director ORI
Frank Brickle, Director ORI
Keith Wheeler, Secretary ORI
Steve Conklin, CFO ORI
Michelle Thompson, CEO ORI
Matthew Wishek, Director ORI