First Principles of Digital Signal Processing

Ward Harriman AE6TY

Pacificon '13

The Basics

Most of us know the basics of electronic circuits:

1) Kirchhoff's current law
2) Ohm's law
3) Thevenin and Norton Equivalents.

Most of us understand how to apply these rules to circuits which have reactive components such as Caps and Inductors.

Today, I'd like to present what I think are the basics of Digital Signal Processing.

Lets get started.

Chapter one:
The Discrete Time Fourier Transform

$$\begin{bmatrix} x_0 & x_1 & \cdots & x_{N-1} \end{bmatrix} \begin{bmatrix} W_N^{0\cdot 0} & W_N^{0\cdot 1} & \cdots & W_N^{0\cdot (N-1)} \\ W_N^{1\cdot 0} & W_N^{1\cdot 1} & \cdots & W_N^{1\cdot (N-1)} \\ \vdots & \vdots & \ddots & \vdots \\ W_N^{(N-1)\cdot 0} & W_N^{(N-1)\cdot 1} & \cdots & W_N^{(N-1)\cdot (N-1)} \end{bmatrix} = \begin{bmatrix} X_0 & X_1 & \cdots & X_{N-1} \end{bmatrix}$$

Implemented using standard linear algebra with the "W" matrix.

For the visual learners....

andrew.nerdnetworks.org/classes/commit/fft-factoring.pdf

Here is a graphic representation.

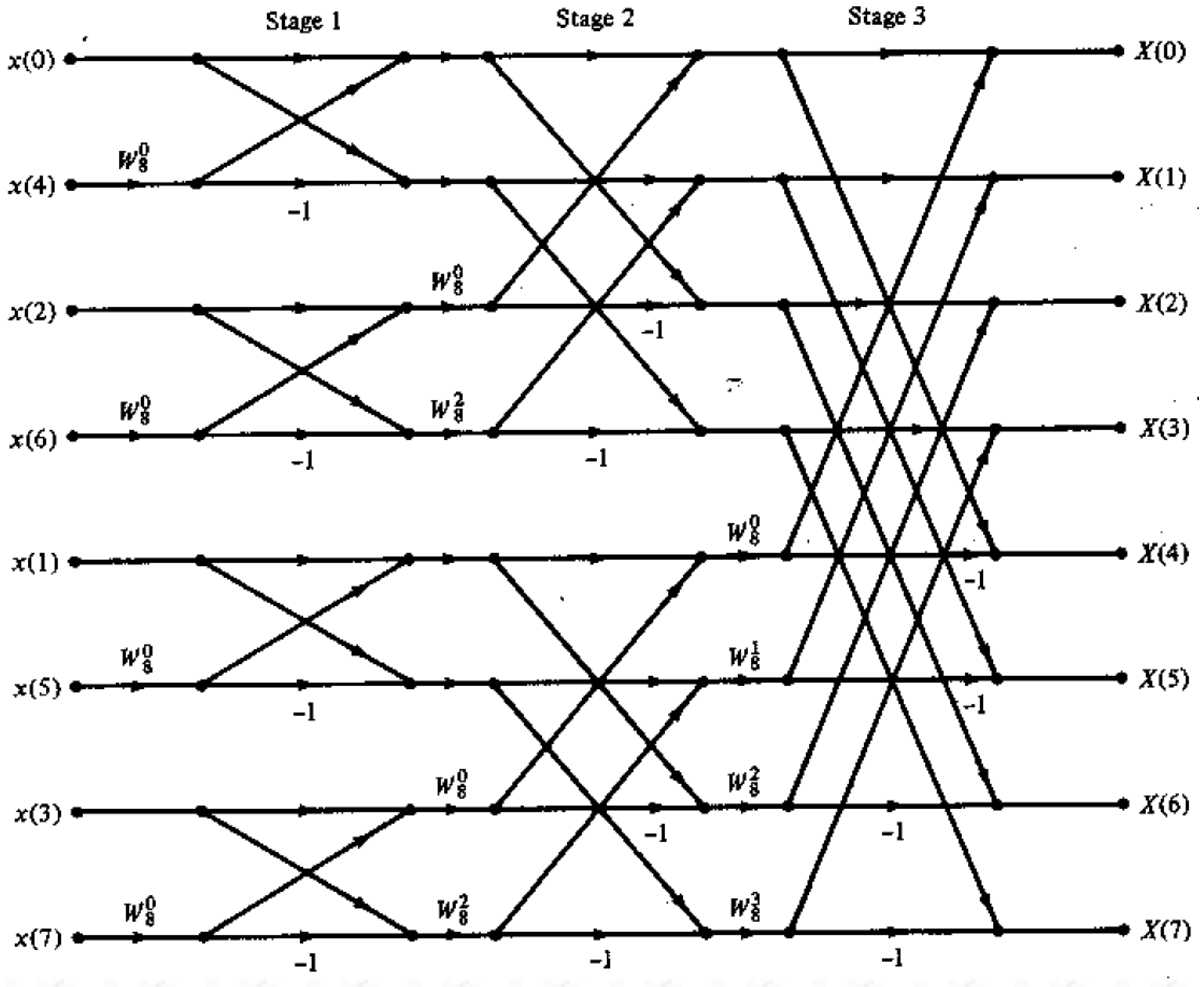
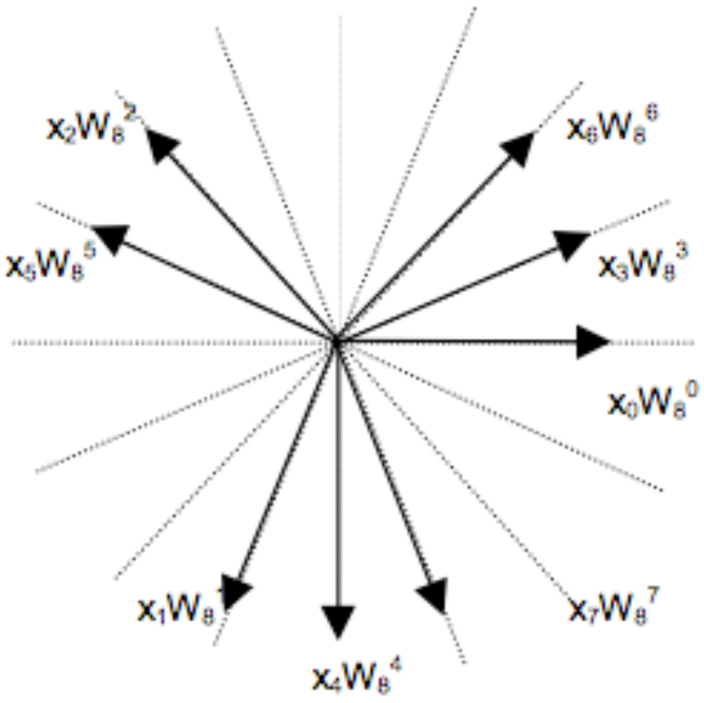


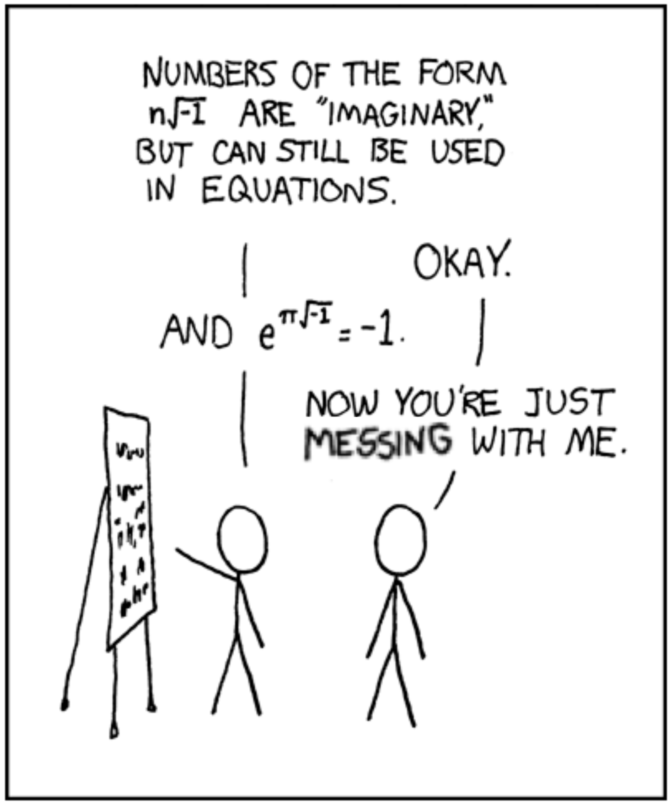
Figure TC.3.3 Eight-point decimation-in-time FFT algorithm.

www.cmlab.csie.ntu.edu.tw/cml/dsp/training/coding/transform/fft.html

Where the W matrix is expressed as rotations of the unit vector in the complex plane



WAIT !!!!!!


NUMBERS OF THE FORM $n\sqrt{-1}$ ARE "IMAGINARY," BUT CAN STILL BE USED IN EQUATIONS.
OKAY.
AND $e^{\pi\sqrt{-1}} = -1$.
NOW YOU'RE JUST MESSING WITH ME.

Yes, I'm Messing With You

Many Digital Signal Processing text books and primers actually start out like my first few slides.

BUT: it doesn't have to be that way.

By understanding a few of the basic principles of DSP you can

1) understand the limitations of the technology
2) appreciate the clever solutions others have found
3) develop a sixth sense for detecting 'unreasonable' claims.

The Basic Steps.

Nearly all Digital Signal Processing applications have these steps:

1) Convert an analog signal to digital.
2) Process the numeric data.
3) Convert the numeric data into analog.

There are fundamental techniques and limitations involved with each step.

Steps 1 and 3 are, strictly speaking, NOT “Digital Signal Processing” but are worth reviewing so... lets start there.

Analog to Digital Conversion

Probably the very first specification in any A/D conversion is the ‘number of bits’...

1) Wide range, 8 bits to 24 bits are quite common.
2) Number Format: for today, we’ll say ‘fixed point’.

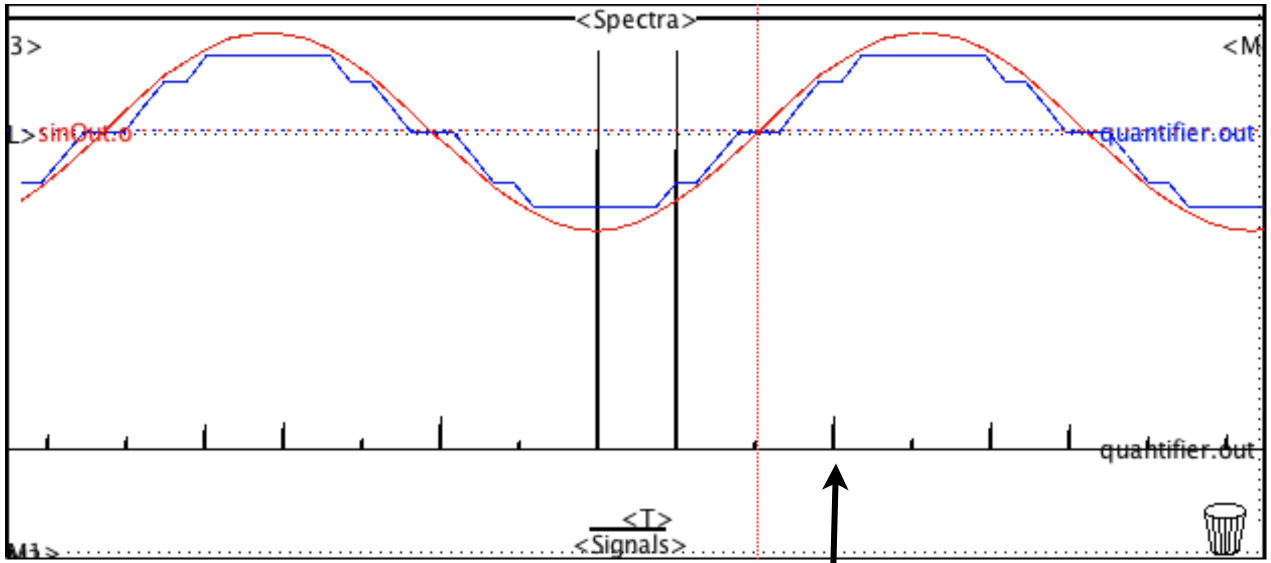
Tradeoffs:

1) more bits generally means more time OR more power.
2) more bits requires faster processor.
3) fewer bits means more ‘quantization noise’.
4) dynamic range of signal can dictate # of bits required.

Quantization Noise

We can examine the impact of “number of bits”...

example: quantization



Notice the noise in the spectrum:

We can play with the number of bits....

Analog to Digital Conversion

And without taking a breath, we ask about “sample rate”.

There is a HUGE range in sample rates, anywhere from 1/day to tens/nanosecond.

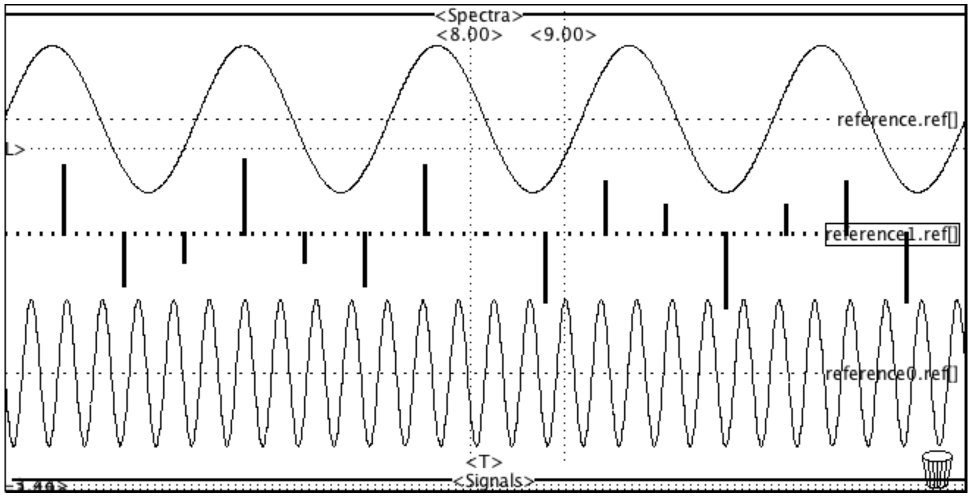
Tradeoffs:

1) Higher rate takes more power.
2) Higher rate requires faster processor.
3) Sample rated dictated by ‘frequency’ of analog signal.

Introducing “Nyquist”.

Basically, Nyquist tells us that you must sample at ‘twice the highest frequency of your input signal’.

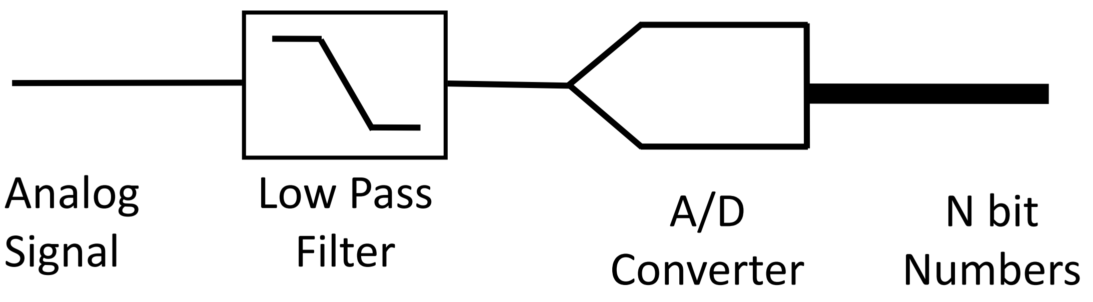
Here’s why... example: Nyquist



Notice: both input waveforms have the same ‘samples’!
this problem is called ‘aliasing’.

Nyquist Filter

To avoid any problems associated with ‘aliasing’ of high frequency signals, an analog filter is generally placed before the A/D conversion:



However:

Since these ‘antialiasing’ filters are generally analog and have some ‘skirt width’, a common tradeoff is:

You need to sample 3 times the highest frequency, not just twice!

Digital to Analog Conversion

Lets jump to the process of Digital to Analog conversion because it encounters many of the same issues as the A/D. Again:

Number of bits:

1) more bits generally means more time OR more power.
2) more bits requires faster processor.
3) fewer bits means more ‘quantization noise’.
4) dynamic range of signal can dictate # of bits required.

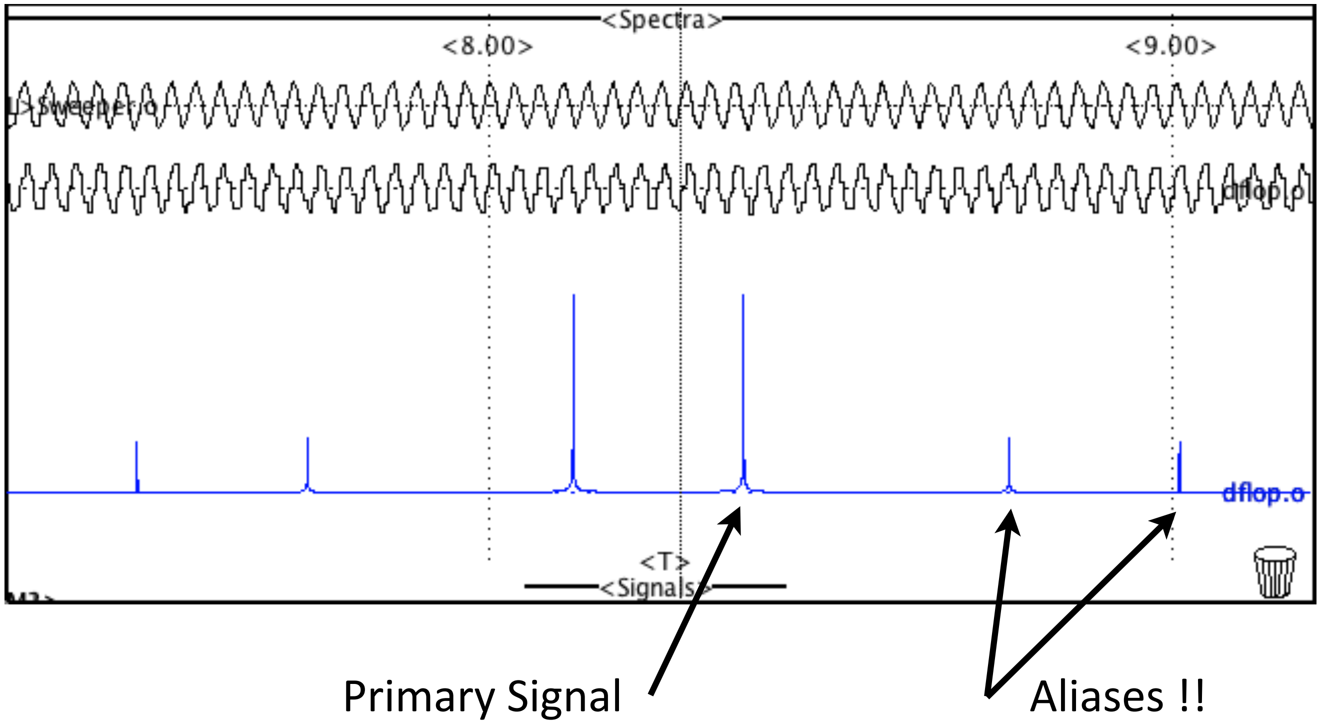
Sample Rate:

1) Higher rate takes more power.
2) Higher rate requires faster processor.
3) Sample rated dictated by ‘frequency’ of analog signal.

One Significant Difference:

In A/D conversion we had to eliminate any alias frequency signal IF IT WAS PRESENT.

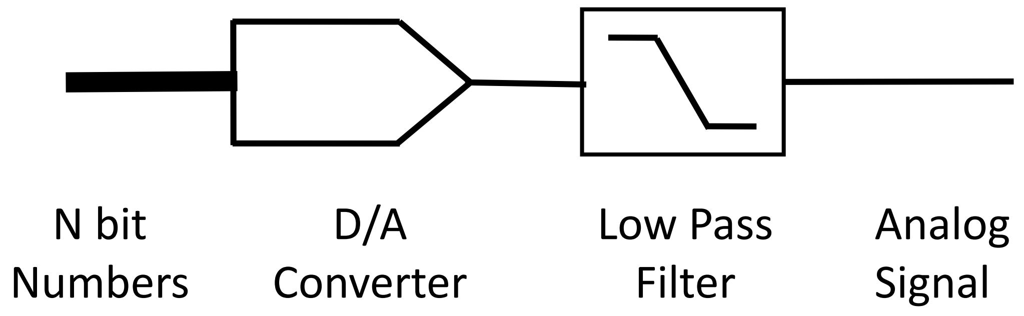
In D/A conversion, the 'alias' frequencies are ALWAYS present. Here is an example of a a 'pure' signal:



EXAMPLE: d2a

Anti Aliasing Filter

So, just as with A/D conversion, D/A conversion must have an antialiasing filter.



And again, because the filters are often analog, we need room for the ‘skirts’ and so, the D/A conversion rate is often 3 times the highest frequency of interest.

The GOOD Stuff: Digital Signal Processing

Having gotten A/D and D/A out of the way, lets talk about the fun things: the actual Signal Processing.

It is often easier to discuss a new topic in the framework of a familiar one. To that end, lets talk about DSP as it applies to filters.

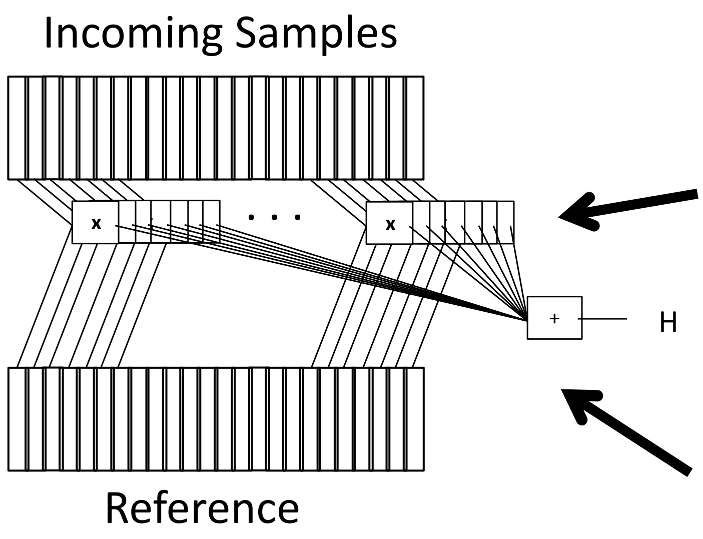
Many DSP algorithms start out by gathering up a number of samples from the input. We'll call those samples the 'input buffer'.

Getting Started with DSP

A Simple Beginning: a very narrow band filter.

To Implement this filter we'll use a DSP technique called "Correlation"

"Correlation" compares the 'incoming' buffer with a 'reference' buffer:



Implementing "Correlation"

Multiply together elements of arrays

and then

Add together the products to get 'H'.

First Principle: Correlation

Correlation has some very important properties.

1) if two signals are the ‘same’ then ‘H’ is large positive.
2) if two signals are exactly opposite, then ‘H’ is large negative.
3) if two signals are exactly ‘out of phase’ then H is ‘0’.
4) if two signals are different then H will be zero.
5) ‘H’ is linear with amplitude of the reference and/or the signal.

example: correlation

1) change phase
2) change amplitude
3) change frequency.

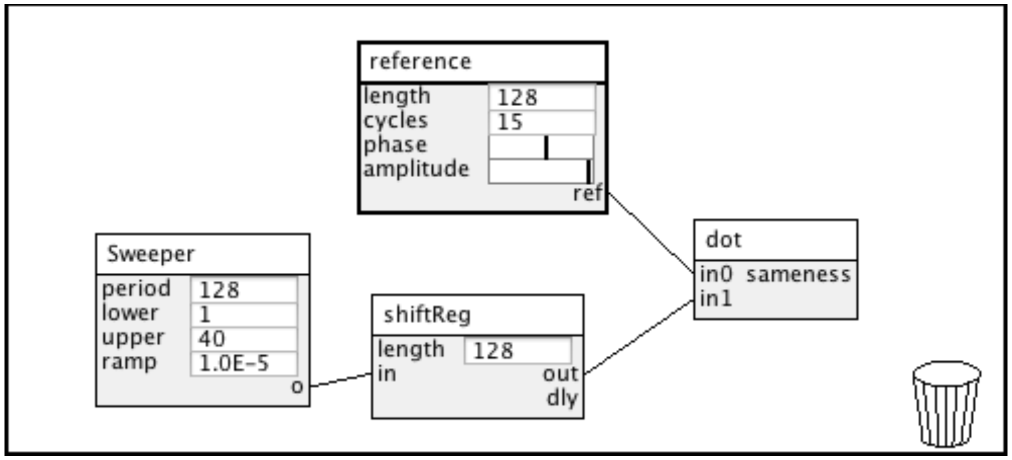
Example: single frequency filter.

Here is a single frequency filter example:

1) The ‘reference’ is 128 samples long with 15 full cycles.
2) the ‘Sweeper’ sweeps frequencies from 1 to 40 cycles
3) the ‘shiftReg’ records the ‘last’ 128 samples.
4) the ‘dot’ does the correlation.

What should we see as the output spectrum? Only the 15 cycles should get through, right?

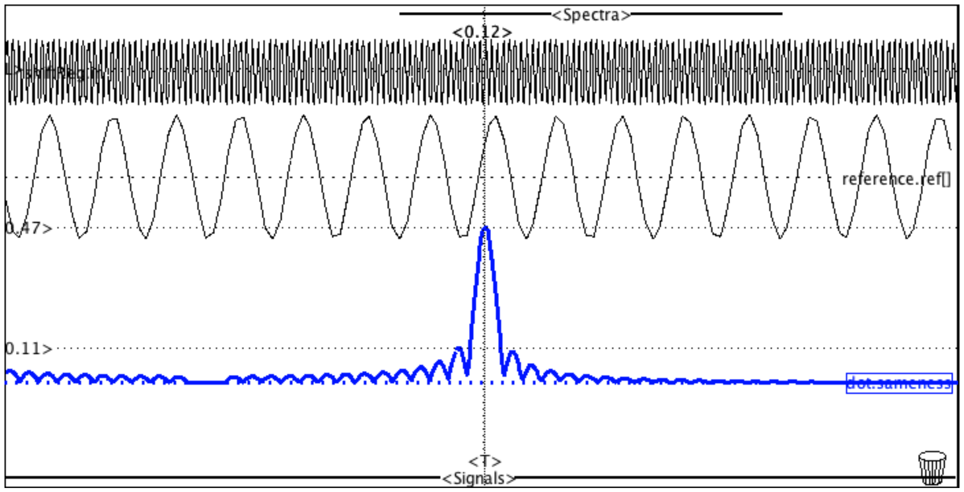
example: singleFreqFilter



Example: single frequency filter.

Unfortunately, NO!

Here is what we see:
There is the main filter frequency AND these pesky little side lobes.



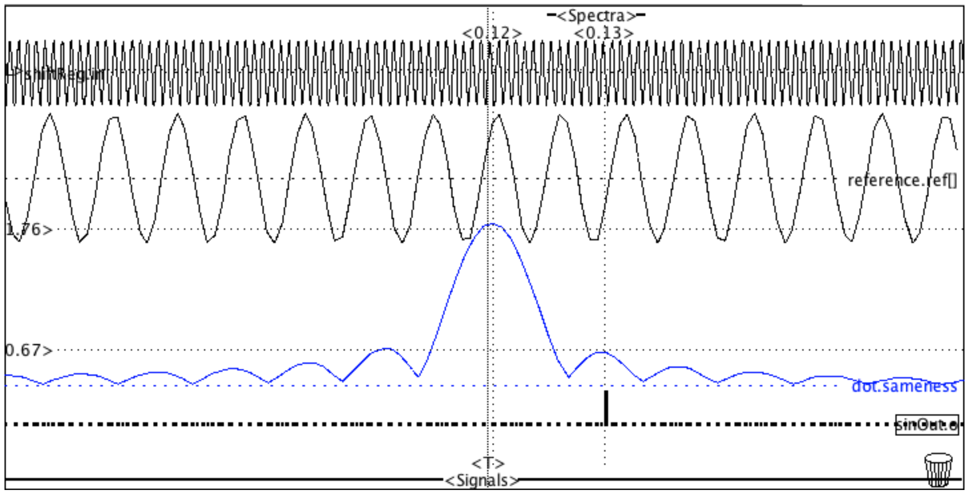
Those side lobes are only down by 12.6 dB or so.

Where did they come from, and how do we get rid of them?

Example: single frequency filter.

Lets look closely at
the output of out filter.

As we might expect,
the output is zero when
the input signal has
exactly N cycles in the
shift register.



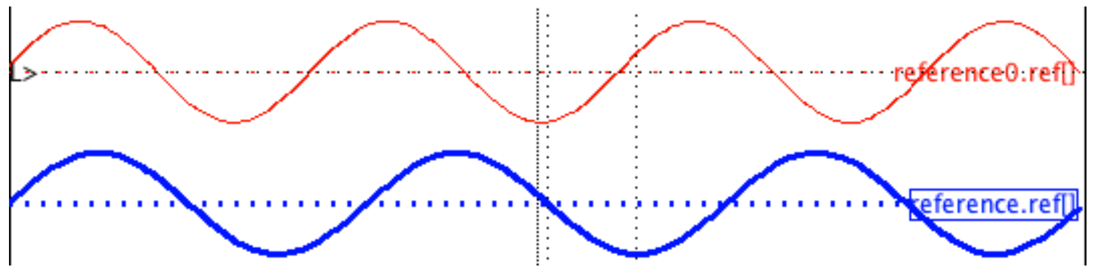
The peaks occur when a signal has between N and N+1 cycles in the shift register.

Principle 2: Spectral Leakage.

Those side lobes are called ‘spectral leakage’ and stem from the violation of a fundamental assumption of correlation.

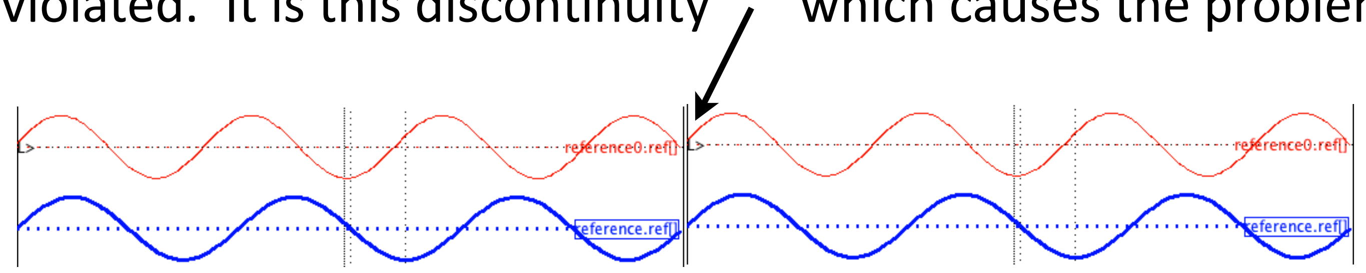
This fundamental assumption is that the input signal is repetitive: if the shift register is 128 samples long then it is assume that every 128 samples the contents of the shift register will repeat. This is NOT the case when the input signal is not an integral number of cycles.

Consider these two waveforms, one 3 cycles and one 3.5 cycles.



Spectral Leakage

If this waveform is placed end/end it is clear that the assumption is violated. It is this discontinuity which causes the problem.

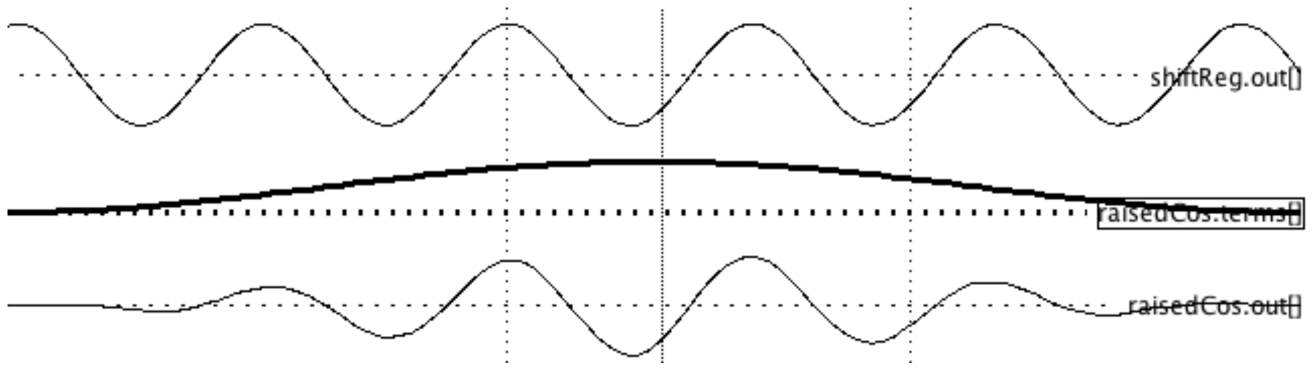


How can this discontinuity be avoided? As it turns out, by brute force, we simply force the contents of the shift register to conform to the assumption as best we can.

This is called "Windowing". There are dozens of 'window' functions but they all do basically the same thing: force the shift register contents to conform.

Principle 3: Windowing

One of the most common window functions is called the ‘raised cosine’. We simply scale the input data by multiplying the original contents as follows:



Multiply each entry in the shift register by the corresponding entry in the ‘raised cosine’ signal. You get the signal at the bottom. The contents of the shift register decay towards zero at each end.... no discontinuity there!

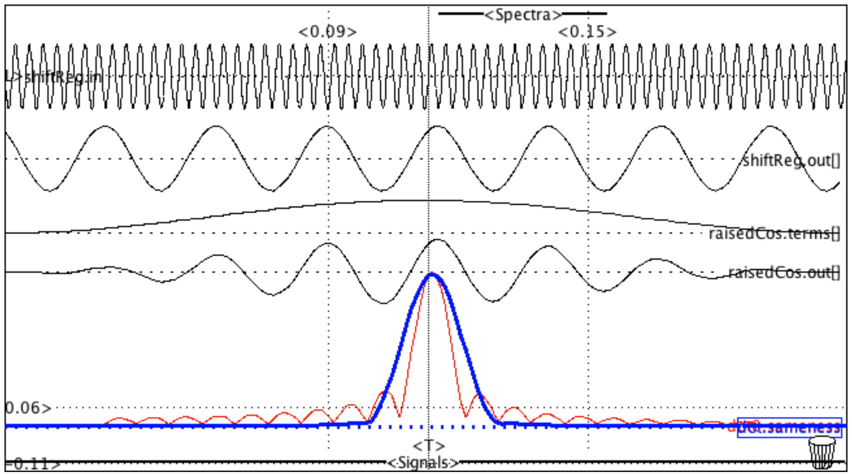
Of course, this is not without cost. The signal has been blurred and so the filter output spectrum will be affected.

Principle 3: Windowing

Here is the result of applying the raised cosine:

example: sweepFreqFilter2

The blue signal is the filter output after applying the raised cosine window. The side lobes are much lower, approximately 33 dB down.



However, the 'skirt' of the filter has gotten wider. It is now about twice as wide as before.

As a general (but not universal) rule, when windowing, lower side lobes result is wider skirts.

Improving Performance.

It would seem that this windowing would have a significant performance impact. Consider:

Before windowing we added up the products of shiftReg[n] * reference[n].

With windowing, we add up shiftReg[n] * raisedCos[n] * reference[n].

BUT: we can see that we can apply the raised cosine to the reference instead and get the same results!!!

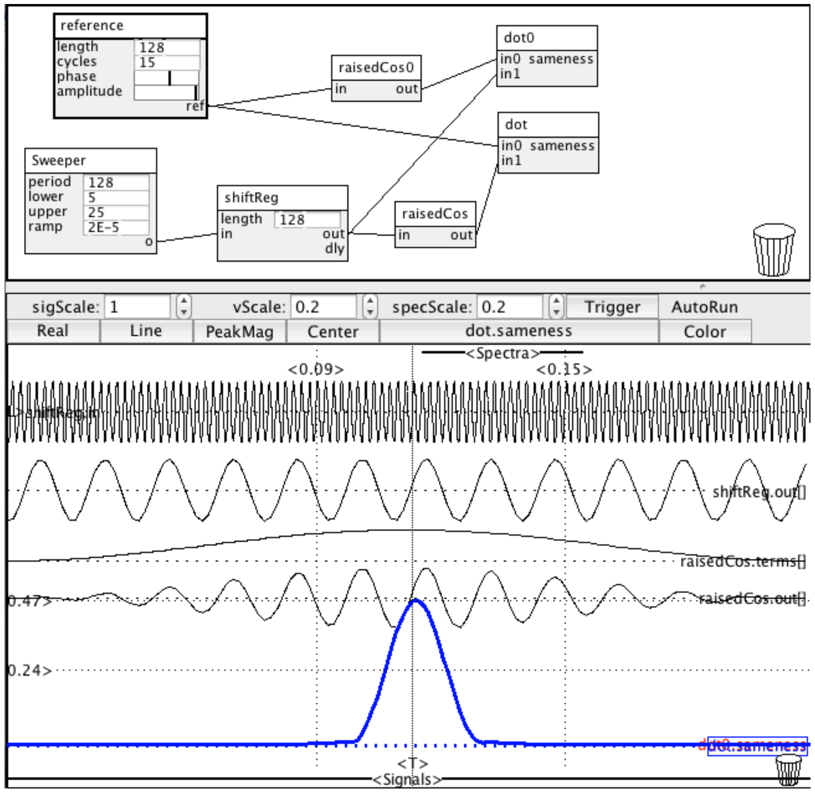
example: sweepFreqFilter3

Notice outputs are the same regardless of where windowing is done.

Windowing Performance.

Here is a screen shot. As we can see, there is no difference and SO, windowing costs compute cycles only while building the reference, not while doing the real work!!!

Notice how the raised cosine is applied to the reference in one path and the sample data in the other. The outputs exactly overlap!


reference
length 128
cycles 15
phase
amplitude

Sweeper
period 128
lower 5
upper 25
ramp 2E-5

shiftReg
length 128

sigScale: 1
vScale: 0.2
specScale: 0.2

Principle 4: Linearity.

Most of us are familiar with the concept of linearity. In essence it says:

If input signal A1 produces output signal B1
and
input signal A2 produces output signal B2

then
A1+A2 will produce B1+B2

This is a hugely valuable property because it means we can build very sophisticated filters.

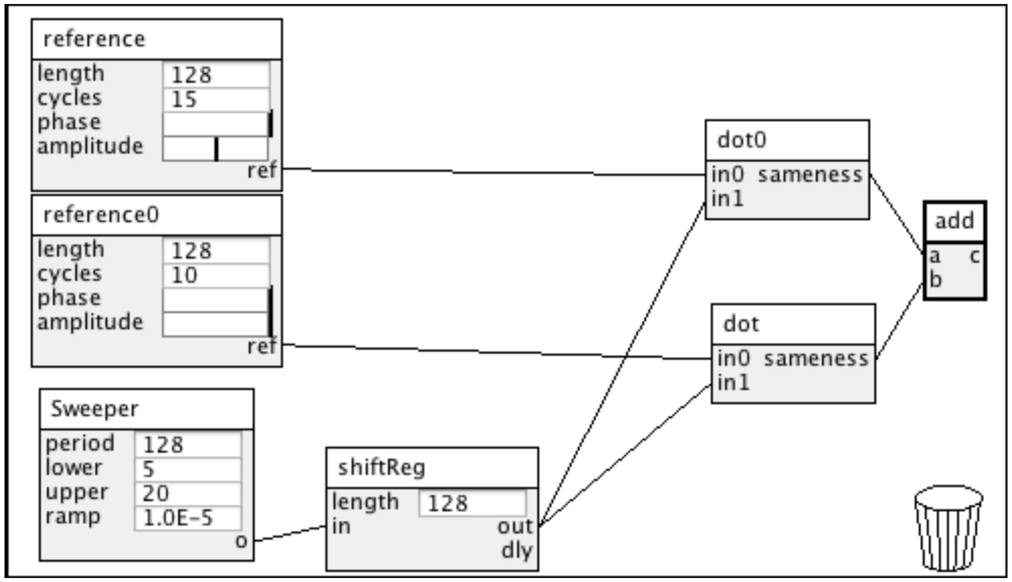
Principle 4: Linearity.

Consider this application... I want my filter to pass two frequencies.
BUT... I want to attenuate one frequency by a factor of 2.
The block diagram could be:

example: twoFreqFilter

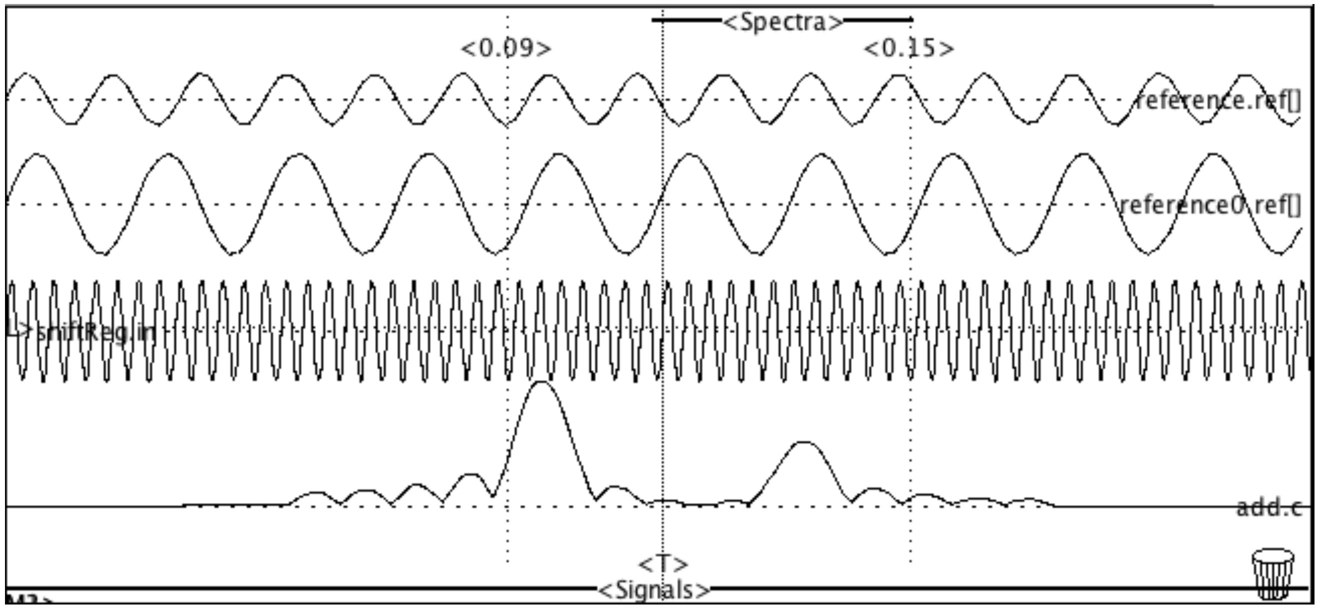
Notice 'reference' has an 'amplitude' of 1/2.

In essence, two filters are provided and their outputs are combined.



Principle 4: Linearity.

Here is the output of this circuit is as expected:



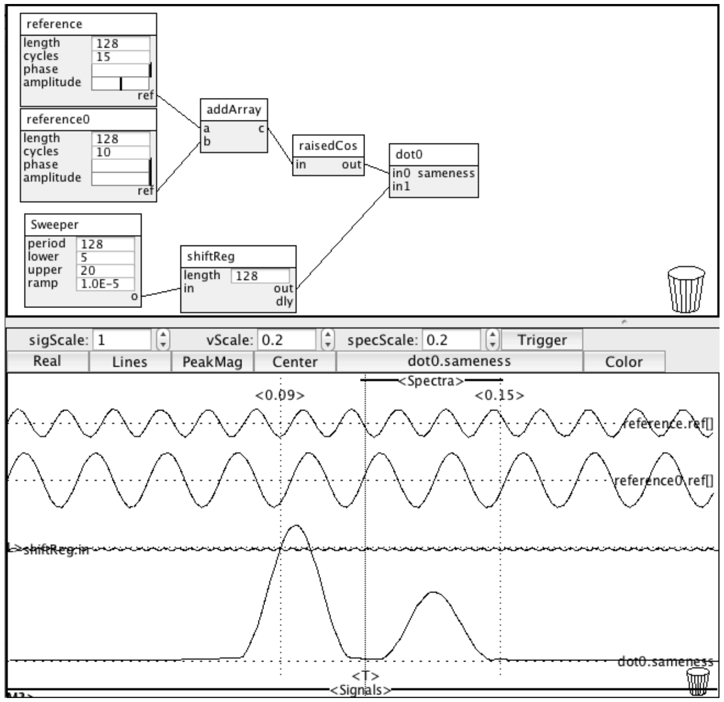
If we wanted to pass a band of N frequencies we could build N filters OR we could use linearity to restructure the filter

Principle 4: Linearity.

Restructured filter with raised cosine added: simply add the two references together and apply the raised cosine and use the result as the 'reference' for the filter.

The implications of this simple technique are HUGE! It means we can build a filter of any bandwidth and adjust the amplitude and phase response of each individual frequency.....

And it doesn't take any more computing than a simple single frequency filter!



reference: length 128, cycles 15

reference0: length 128, cycles 10

Sweeper: period 128, lower 5, upper 20, ramp 1.0E-5

shiftReg: length 128

sigScale: 1 vScale: 0.2 specScale: 0.2

Principle 5: Spectral Resolution

Lets change gears.

How can we determine the ‘spectral resolution’ of our DSP system?

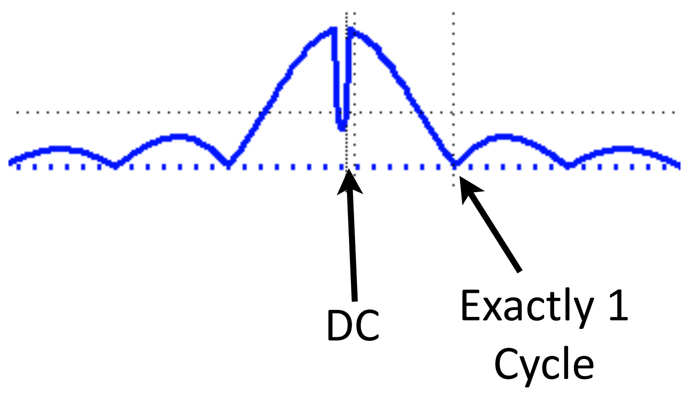
Lets think back to our filter example. The simple filter (no windowing) had the best skirt. (Not the best stop band!)

This skirt represents a measure of the spectral resolution of a particular DSP algorithm.

So how wide, in frequency, was this skirt?

Principle 5: Spectral Resolution

Here is a plot of a filter for 'DC'.



As can be seen, the skirt width is whatever frequency has exactly 1 cycle in the shift register.

This means, to a good approximation, the skirt width is controlled by HOW LONG (in time) the shift register is.

Principle 5: Spectral Resolution

For example, if the shift register contains 1 millisecond of the input signal then this filter skirt will be 1 kH wide.

If the shift register contains 10 milliseconds, the filter skirt will be 100 Hz wide.

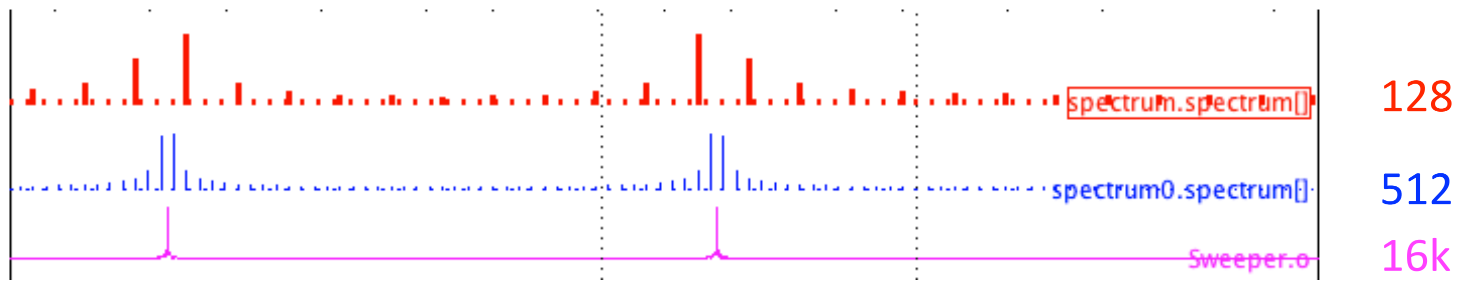
There are some important implications of this.

1) There is no substitute for ‘time’.
   If you want to measure the frequency of a signal to high resolution, you have to take your time.
2) The ‘Sample Rate’ has nothing to do with it.
   Sampling a signal at a higher rate does NOT improve frequency resolution!

Principle 5: Spectral Resolution

Here is a plot showing the spectrum of a signal with different 'shift register' lengths but the same sample rate.

example: spectralResolution



As can be seen, spectral resolution is a function of how LONG, in time, the signal has been observed.

Wrap Up

In this presentation:

1) We started out reviewing basic A/D and D/A conversion.
    1) number of bits.
    2) Sample rate... Nyquist and AntiAliasing filters.

2) We then examined some “First Principles of DSP”...
    1) Correlation
    2) Spectral Leakage
    3) Windowing
    4) linearity
    5) Spectral Resolution

We didn’t talk about:

Notice that we DIDN’T Talk about:

1) Fourier Transforms
2) Laplace and Z transforms
3) Finite Impulse Response Filters

Summary and Thank You

I hope this presentation has:

Given you a chance to review basic A/D D/A issues

Introduced some of the basics of DSP processing.

Provided a foundation for further investigation.

Thank you.

With TIme: A/D Sensitivity and Noise

One of the counter-intuitive properties of Digital Signal Processing is that noise is NOT ALWAYS your enemy.

Indeed, it can improve the sensitivity of your A/D converter.

To demonstrate this: 1noiseInjection

Notice how adding some noise will bring out a signal which was otherwise undetected:

With Time: Over Sampling and Decimation

One of the things which perplexed me for a long time was the fact that one can use oversampling and decimation to improve sensitivity and dynamic range of and A/D converter.

How can this be?

In essence, one simply samples N times and then averages those N samples (overSample by N, decimate by N).

Here is an example of N = 16. example: decimation

NOTE:
- effectively adds Log2(N) bits to A/D conversion.
- increases noise by a factor Sqrt(N) (scale
- Can decrease quantization errors (adjust for harmonic distortion)