# MOUNTAIN SPARK GAPS

**NPARC—The Radio Club for the Watchung Mountain Area**

**Website: http://www.nparc.org**
**Club Calls: N2XJ, W2FMI**
**Facebook: New Providence Amateur Radio Club ((NPARC)**

VOLUME 53 NO. 1 January 2018

## UPCOMING EVENTS

**Auction**
**2/24/2018 1:30 - 4:00 PM**
**Clean out the extra stuff in the shack and make some money.**

## <u>Regular Meetings</u>

2/12 & 2/26
Monday 7:30

## Meeting Schedule

**Regular Meeting:** 7:30—9:00 PM
**2nd Monday of each month** at the
NP Senior & Adult Center
15 East Forth Street
New Providence

**Informal Meeting:** 7:30—9:00 PM
**4th Monday of each month**
**Same location**
**Everyone is Welcome**
If a normal meeting night is a holiday, we usually meet the following night.
Call one of the contacts below
or check the web site

## Club Officers for 2018

President: W2PTP Paul Wolfmeyer
201-406-6914
Vice President:K2GLS Bob Willis
973-543-2454
Secretary: K2AL: Al Hanzl
908-872-5021
Treasurer: K2YG Dave Barr
908-277-4283
Activities: KA2MPG Brian Lynch
973-738-7322

## On the Air Activities

Club Operating Frequency
145.750 MHz FM Simplex

Sunday Night Phone Net
Murray Hill Repeater (W2LI) at 9:00 PM
Transmit on 147.855 MHz
With PL tone of 141.3 Hz
Receive on 147.255 MHz
Net Control K2AL

Digital Net
First & Third Mondays 9 PM
28,084 — 28,086
Will be using PSK and RTTY
Net control K2YG

## Club Internet Address

Website: http://www.nparc.org
Webmaster KC2WUF David Bean
Reflector: nparc@mailman.qth.net
Contact K2UI, Jim

## MOUNTAIN SPARK GAPS

Published Monthly by NPARC, Inc.
The Watchung Mountain Area Radio Club
P.O. Box 813
New Providence, NJ 07974
©NPARC 2010 All Rights Reserved
Editor: K2EZR Frank McAneny
Contributing Editors:
WB2QOQ Rick Anderson
W2PTP Paul Wolfmeyer
K2UI Jim Stekas

Climatological Data for New Providence for December 2017

The following information is provided by Rick, WB2QOQ, who has been recording daily weather events at his station for the past 36 years.

**TEMPERATURE -**
Maximum temperature this December, 62 deg. F (December 5)
Last December(2016) maximum was 56 deg. F.
Average Maximum temperature this December, 38.7 deg. F
Minimum temperature this December, +6 deg. F (December 31)
Last December(2016) minimum was 13 deg. F.
Average Minimum temperature this December, 24.9 deg. F
Minimum diurnal temperature range, 5 deg. (33-28) 12/9
Maximum diurnal temperature range, 22 deg. (55-33) 12/6

Average temperature this December, 31.8 deg. F
Average temperature last December, 34.6 deg. F

**PRECIPITATION -**
Total precipitation this December - 1.62″ rain/melted snow; 7.5″ snow
Total precipitation last December - 3.23″ rain/melted snow; 2.3″ snow

Maximum one day precip. event this December - December 23, 0.57″ rain; Dec. 9, 4.0″ snow.

Measurable rain fell on 5 days this December; Measurable snow fell on 5 days.
YTD Precipitation - 44.86″

======================
Rick Anderson
1/5/18
243 Mountain Ave.
New Providence, NJ
(908)464-8911
rick243@comcast.net
**Lat = 40 degrees, 41.7 minutes North**
**Long = 74 degrees, 23.4 minutes West**
Elevation: 380 ft.
CoCoRaHS Network Station #NJ-UN-10

# ***Direct Sampling SDR Receiver : FPGA***

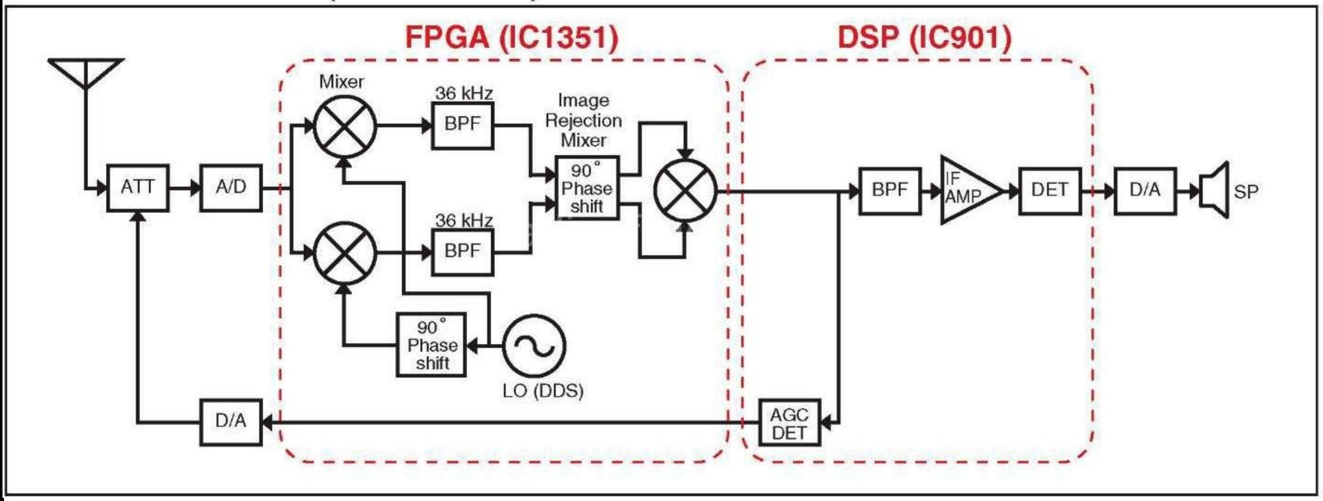
*Jim Stekas - K2UI*

Last month we discussed the A/D converter in the front end of a direct conversion receiver. For the A/D to do its job two things are essential:

1. for weak signals the gain of the RF front end must be adjusted to allow the noise floor to toggle the lower order bits of the A/D, and
2. for strong signals, the AGC must reduce the gain to prevent clipping.

This month we continue our discussion with the digital processing of the A/D output that occurs in the FPGA using the IC-7300 receiver (figure below) as an example. Note that the block diagram of the FPGA processing looks no different from the mixer and IF stages of an analog radio using the phasing method of SSB detection. Sections that follow discuss possible implementations of the FPGA processing blocks, and not necessarily what Icom implemented in the IC-7300.

• FPGA BLOCK DIAGRAM (Receive circuits)



## Field Programmable Gate Array

A Field Programmable Gate Array (FPGA) is a chip with a large array of building bocks for constructing digital circuits: logic gates, flip-flops, registers, look up tables, adders, multipliers, etc. The configuration and inter-connection of these elements is defined in a “BIT” file that loaded from an EPROM at power up. The BIT file is generated by compiling VHDL (or Verilog) source code in much the same way that a Windows EXE file is generated by compiling a C-language program. Can Joe Q. Ham do his own FPGA programming? The good news is that it's legal in all 50 states. The bad news is the learning curve is very steep and the tools are expensive. (More about this later.)

## Clock

One critical element that is omitted from the block diagram is the CLK (clock) that drives the FPGA and A/D. Typically the a clock generator chip is used to generate the clock from a 10MHz reference oscillator. In the IC-7300 the A/D runs at 125MHz (approx) so it is reasonable to expect the FPGA to run off the same clock. A more typical arrangement would be to clock the FPGA at 250MHz and have the FPGA generate the 125MHz A/D sample clock with a divide-by-two circuit. This allows the

FPGA to do twice the computing it would at 125MHz.

**Local Oscillator**

The local oscillator (LO) is a Direct Digital Synthesizer (DDS) that computes $\cos(\omega_{LO} t)$ every 8nsec or 125M times per second. The DDS uses a lookup table to determine values of $\cos(\phi)$ and $\sin(\phi)$. A table of 360 entries would allow cos/sin values to be stored for every degree of phase, which is a reasonably fine resolution. If greater than 1$^o$ precision is needed a larger table could be used in conjunction with interpolation using a bit of high school trig. For a phase $\phi$ close to the nth table entry, $\phi_n$, to very good approximation (using radians):

$$\cos(\phi) \approx \cos_n - (\phi - \phi_n) \cdot \sin_n \quad \text{and} \quad \sin(\phi) \approx \sin_n + (\phi - \phi_n) \cdot \cos_n$$

where $\cos_n$ and $\sin_n$ are lookup table entries. Since the DDS is keeping track of both cos() and sin() the 90$^o$ phase shift block is not actually needed as it would be if the LO was an analog oscillator. We will assume the cos() signal is sent to the upper mixer and the sin() to the lower mixer.

The simple DDS described above might be fine for ham gear but it wouldn't cut it in a piece of Tektronix test gear because truncation/rounding errors would introduce phase noise to the LO signal, as well as weak “spurs”. Additional trickery is needed to produce a lab quality LO signal.

**Digital Mixers**

The digital mixer blocks are trivially implemented by simple integer multiplication of the input signal samples by the LO sample values. Since the multiplication is a perfect mathematical operation there is no leakage of input signals or LO to the output of the mixer and no second or third order products typical of analog mixers. Thus the digital mixers are essentially “perfect” double balanced mixers so long as the multiplication doesn't overflow (i.e. clip). Internally, FPGA digital signals will typically be 16-bit signed integers, so the product of two signals can result in a 32-bit signed result in the worst case. This is commonly called “bit-growth” and is something DSP designers lose a lot of sleep over. Common practice is to predict the RMS signal at the multiplier output (through modeling and simulation) and select the most significant 16 bits that actually capture the signal without clipping. If 16 bits aren't enough it may be necessary to pass 20 or 24 bits to downstream stages. Ouch!

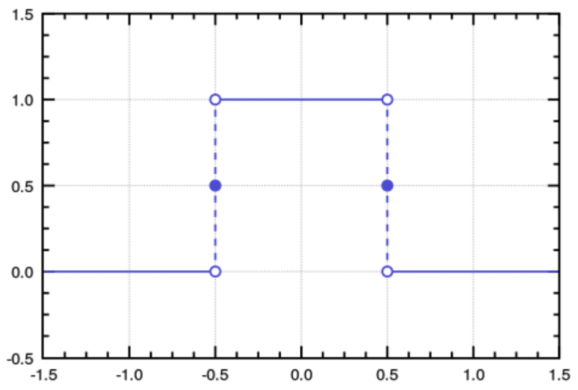
By convention, the I (in-phase) signal comes from the mixer receiving the cos() version of the LO, the upper mixer in this case. The Q (quadrature) signal comes from the lower mixer which receives the sin() version of the LO.

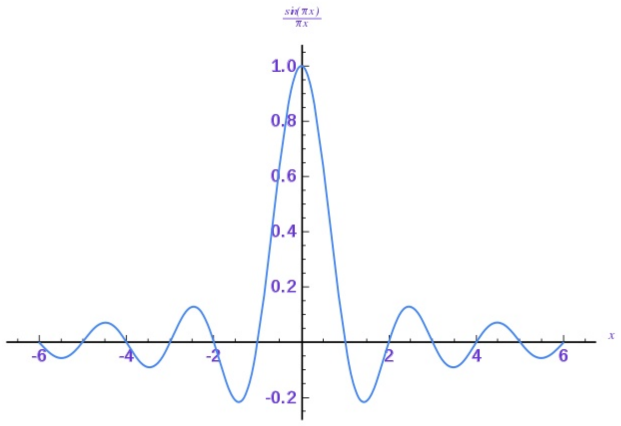
**Bandpass Filter**

Let's assume we are working QRP CW on 14.030 MHz. We could set our LO at 14.029300 which would result in a 14.030 CW signal being converted to +700Hz audio at the mixer output. But everything else in the HF spectrum would also be present at the mixer output so we need to filter out the slice of spectrum that we want. In the IC-7300 that means a 36kHz slice. In the quadrature I/Q world of SDR a 36kHz BPF (band pass filter) is actually implemented with two 18kHz LPFs (low pass filters), one for I and the other for Q. This passes frequencies from -18 to +18 kHz, giving us our 36kHz bandwidth. What's the difference between +1kHz and -1kHz? The answer is that the

-1kHz sits on the LSB side of zero beat and the +1kHz on the USB side. If we tune up 100Hz the signals would become +900Hz and -1100Hz.

Suppose we want our LPF to be a perfect brick wall filter, i.e. one that passes signals from 0 to 18kHz with no loss and blocks all higher frequencies (figure at right).





If we put a short impulse through such a filter and observed the output on an oscilloscope we would see that the "impulse response" of the filter would have the shape of a $sinc()$ function (left):

$$sinc(2f_{max}\cdot t) \;=\; \frac{\sin(2\pi f_{max}\cdot t)}{2\pi f_{max}\cdot t}$$

where $f_{max}=18\,kHz$

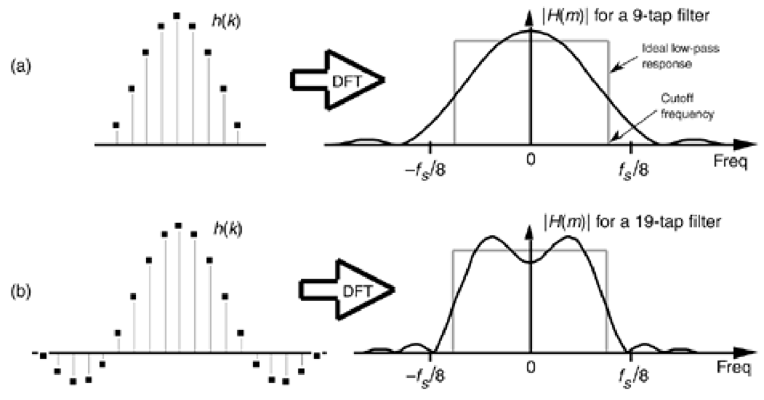
On the FPGA the most common digital filter used is the Finite Impulse Response (FIR) filter because they are linear, simple and well behaved. The general FIR is defined by an series $h(k)$ which generates output samples $y(n)$ from input samples $x(n)$ according to the formula:

$$y(n)=\sum_k x(n-k)\cdot h(k) \quad .$$

Note that if the filter input is an impulse $x(0)=1$ and $x(\neq 0)=0$ the output $y(k)=h(k)$ , so $h(k)$ is actually the impulse response of the filter.

If we copied the values of $sinc(2f_{max}t_k)$ into $h(k)$ we would have a perfect brick wall FIR filter. The problem with this is that $sinc()$ falls off as $1/t$ but never actually gets to zero. So it isn't possible to create a FIR that implements a perfect LPF because $h(k)$ would need to be infinitely long . To fix this we can apply a "window" function , $w(t)$ , that goes to zero outside $-T<t<T$ . The result is a filter $h(k)=w(t_k)\cdot sinc(2f_{max}\cdot t_k)$ that is finite in length and approximates an ideal LPF.

The cutoff time T and the shape of $w(t)$ will effect the steepness of the transition band, stop band attenuation, ripple, etc. The figure below shows the frequency response of filters using different cutoff times. The longer cutoff time (figure b) gives a response that more closely approximates the ideal. (In DSP jargon, each value of $h(k)$ is called a "tap".)



## Decimation

Once we have applied our 18kHz LPF the signal bandwidth is reduced and we can reduce the sampling rate to 36KHz (or greater.) We can "decimate" the LPF output by dropping 2499 out of every 2500 samples to end up with a 50kHz sample rate, plenty good enough. In an actual implementation we would "decimate" by only computing the values of every 2500th sample and not waste resources computing samples that would be dropped.

When we pick a value of T for our filter we need to be sure we allow a significant number of sinc() oscillations, say 10. For our 18kHz LPF this gives $T \approx 0.5\,msec$ which means we would need $h()$ to have a length of at least 60,000 "taps" to filter the raw 125MHz signal. A much more efficient solution is to generate the final IF signal using multiple stages of filtering and decimation.

Often, the first filter in the filter/decimate chain is a Cascaded Integrator–Comb (CIC) filter. The CIC is a clever filter design that is very efficient because it uses no multiplications, only addition and subtraction (i.e all the $h(k) = -1, 0 \text{ or } 1$ ). On the down side, the CIC frequency response is not even close to an ideal LPF so it is generally followed by a FIR filter to flatten the passband and improve the stopband performance.

## Image Rejection Filter

When operating SSB, the image rejection filter is used to combine I and Q to cancel the unwanted sideband. This block is needed because the final stages of the receiver are analog.

**<u>FPGA Programming</u>**

So, is it time to dive into FPGA programming?   Well …  if you have to ask the answer is probably no.

First, ask yourself what you want to do on the FPGA.  If you want to mine bitcoins or build a wideband phased array then you have a good application for an FPGA.   Or if you are sick of programming Arduinos and want to move up to the big leagues, that's also a good reason.

But before diving in,  consider  all of the exciting new digital modes of the last 20 years, especially those from K1JT, that were developed on and for the PC.  None of the them use or require any specialized FPGA code.