Software Defined Radios – The Future is Now

By:
Bob McGwier, N4HY
Gerald Youngblood, AC5OG
Eric Wachsmann, FlexRadio Systems Software Engineer

Background

A Software Defined Radio (SDR) is a radio in which all modulation and demodulation functions are defined, and therefore configurable, through software. This creates tremendous flexibility to improve and adapt the capabilities of the radio over time without changing the hardware. The potential for amateur radio experimentation is virtually limitless in terms of performance improvement and the introduction of new operating modes.

The idea for the SDR-1000 Software Defined Radio was formed about six years ago while observing PSK31 running on a PC and sound card. Effectively, PSK31 uses the sound card and PC as a Digital Signal Processor (DSP) to perform modulation and demodulation of a digital signal. It became clear that a phasing-type transceiver could be built using the stereo inputs of the sound card for the in-phase (I) and quadrature (Q) signals. Four years and many hundreds of hours of study resulted a working transceiver that was described in the four part *QEX* series, "A Software Defined Radio for the Masses[1]." Interest generated by the articles was so strong that a decision was made to begin shipping the radio as a product in April of 2003. The articles, as well as complete information on the SDR-1000, are available on the FlexRadio Systems website at www.flex-radio.com.

The SDR-1000 ships with open source software written in Visual Basic 6, allowing users to modify and improve the code. Hams from all over the world have contributed to the enhancement of the SDRConsole code including both user interface improvements and advanced DSP code. Furthermore, a number of colleges and universities are using the SDR-1000 as part of their engineering curriculum.

The SDR-1000 continues to evolve based on input from the amateur radio community. This paper will focus on hardware and software enhancements that are in process and will be made available in the first half of 2004. The writing is a combined effort by Bob McGwier, N4HY, Gerald Youngblood, AC5OG, and Eric Wachsmann, FlexRadio software engineer.

SDR-1000 and SDRConsole Architecture – Gerald

As stated earlier, the SDR-1000 was described in some detail in the *QEX* series (endnote 1). The initial product consisted of a three-board set as seen in Figure 1. Recently, the enclosure shown in Figure 2 was added to allow a number of enhancement products to be added to the radio.

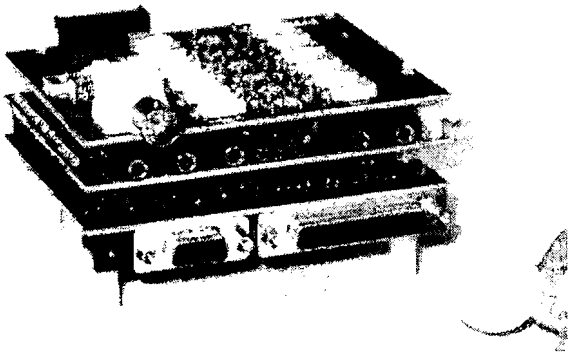

Figure 1 – SDR-1000 Board Set

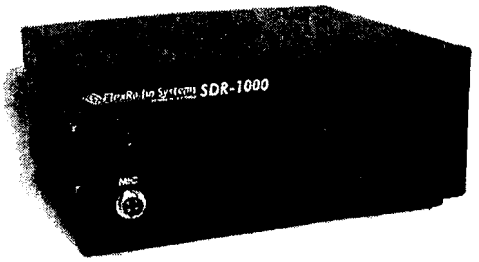

Figure 2 – SDR-1000 Enclosure

The SDR-1000 incorporates a novel Quadrature Sampling Detector (QSD), which offers exceptional dynamic range as well as performing the function of a high Q tracking filter. Figure 3 illustrates a simplified version of the detector. It may be thought of as a rotary switch that rotates at the carrier frequency rate. Each of the four capacitors sample (or integrate) the RF signal for 25% of the carrier period at intervals of 0°, 90°, 180°, and 270°. By differentially summing the 0° and 180° signals and the 90° and 270° signals, we can generate the in-phase (I) and quadrature (Q) signals respectively. With I and Q, we can demodulate or modulate any type of signal. Not only is the carrier mixed to DC by the sampling process, the RC network formed by the antenna impedance and the sampling capacitors forms a commutating filter with a bandwidth of 1/(pi*n*R*C) (where n is equal to the number of sampling capacitors).

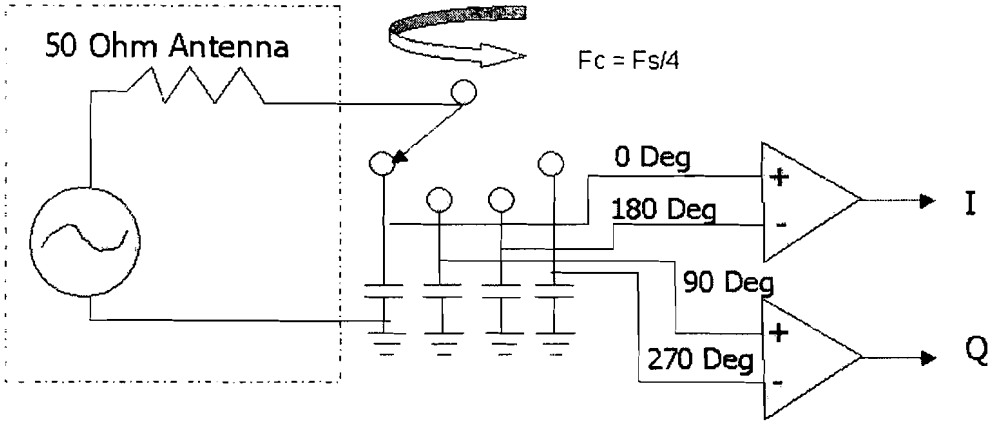

Figure 3 – Single Balanced Quadrature Sampling Detector (QSD)

The SDR-1000 uses a dual 4:1 MUX/DEMUX in a double-balanced QSD circuit that offers a 6dB improvement in large signal handling over the single balanced circuit shown in Figure 3. Using 5V parts, the QSD is capable of handling 10Vpp differential signals before going into compression. The double-balanced circuit also helps to suppress even-order harmonics. This large-signal handling capability allows more flexibility in gain distribution that is traditionally found in direct conversion systems. Gain can be placed in front of the QSD to improve the noise figure and reduce local oscillator radiation without significantly compromising large-signal handling capability.

While analog radios are properly characterized for distortion using third order (IP3) dynamic range, SDRs may not be adequately characterized in the same way. With a properly designed SDR, the radio will be highly linear up to the point of Analog to Digital Converter (ADC) full-scale saturation. When saturation is reached, the signal will be completely distorted and unusable. This means that third order products may not be detectable right up to about 1-2dB under ADC saturation. The full-scale voltage limit of the ADC will therefore set the maximum signal without distortion in a SDR. This may be as high as 4-5Vpp on some converters.

Because the SDR-1000 uses an offset baseband IF of 11KHz, it is possible to avoid many of the issues that have traditionally plagued direct conversion receivers. Above 1KHz, most of the 1/f noise, AC hum, and microphonic noise goes away and the dynamic range of the sound card greatly increases. Once the quantizing level of the ADC has been reached due to the total noise voltage in the filter bandwidth, it can actually resolve signals over a wider dynamic range than the 6dB per bit indicated by the converters resolution. For example, a high quality 16-bit converter has been measured to have a two tone, third order dynamic range of 90dB (RFE installed). Therefore, the dynamic range of the ADC will have the greatest effect on the dynamic range of the receiver in most configurations. Figure 4 illustrates two tone dynamic range using –20dBm tones at 1KHz spacing. Spurs may be seen approximately 95dB down from the fundamental tones.

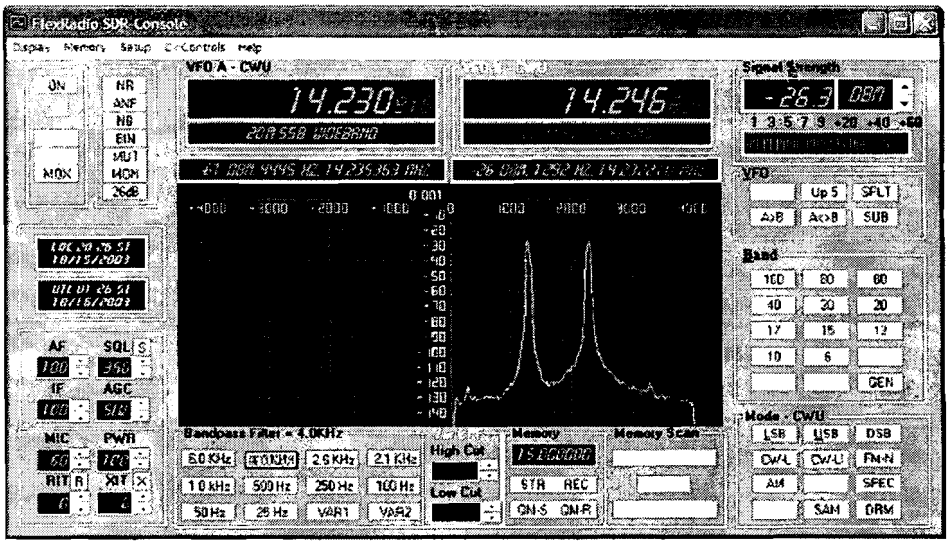

Figure 4 – Two Tone IMD Dynamic Range

The wide linear range of audio ADCs found in the better sound cards allows for some very interesting capabilities. First, it eliminates the need for analog AGC. This means that AGC can be performed digitally after the final filter, thereby greatly reducing the effects of strong adjacent signals. The effect is to remove the "pumping" of the AGC that is characteristic of analog AGC systems.

Further, by using double precision floating point values and fast convolution filtering in the frequency domain, we can achieve bandpass filter shape factors that exceed 1.05:1 (500Hz BW). A 2048-tap filter with 4096-bin FFT achieves stop band attenuation in excess of 120dB within 300 Hz of the 3dB cutoff frequency. Figures 5 and 6 demonstrate the frequency response

characteristics of the 500Hz and 2.7KHz filters respectively. A description of the digital AGC system and fast convolution filters is provided in Part 3 of the *QEX* article series.

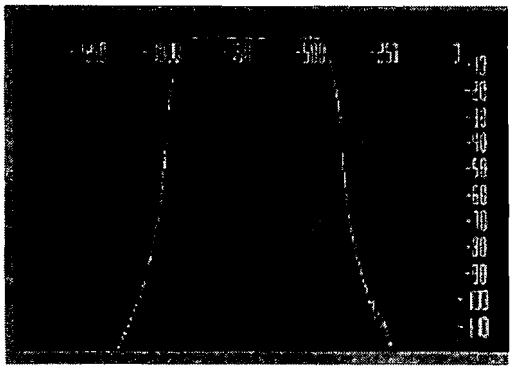

Figure 5 – 500Hz Filter

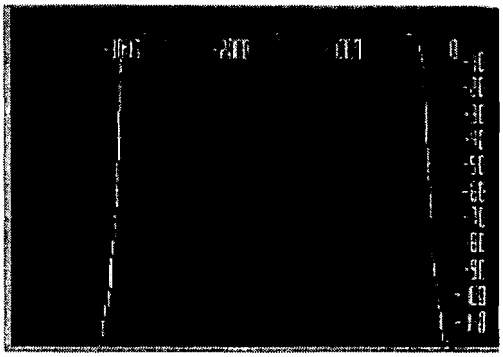

Figure 6 – 2.7KHz Filter

On the modulation side, we can also use the magic of DSP to do feed-forward compression of the audio signal to greatly improve average transmitted power without excessive distortion. The SDR-1000 uses a method of feed-forward speech compression wherein the gain is turned down quickly when the input signal is too large, but increases slowly if the signal drops off or ceases. This prevents the gain from increasing quickly between words. The net effect is similar to that of a good RF clipper without injecting distortion. VK6APH contributed the SDRConsole code for the speech compressor based on the algorithm in Marvin E. Frerking's book, "Digital Signal Processing in Communications Systems[2]."

Another capability of the SDR-1000 is that it also functions as a high dynamic range spectrum analyzer. The SDRConsole, as seen in Figure 4 above, may be calibrated with a signal generator so that its spectrum display and digital readouts display the actual signal levels over a frequency range equal to just under the sampling rate of the sound card (typically 40KHz). Use of quadrature signals doubles the effective sampling rate over that of a single channel. As stated earlier, signals may be measured over a range of 120dB using a high quality sound card.

SDR-1000 Hardware Enhancements

A number of new hardware add-on products extend the radio's capabilities and performance. Now shipping are a new RF Expansion board (RFE) and the Down East Microwave 2M transverter IF for the SDR-1000. A 100W PEP integrated linear amplifier and an automatic antenna-tuning unit will be added in the fall.

The SDR-1000 was designed for general coverage reception up to 65MHz. This requires compromise on the input band pass filters to minimize system cost. The RFE board adds 5th order low pass filters for each amateur band to enhance third harmonic rejection. Further, it adds a noise figure preamplifier ahead of the QSD that allows a 3dB receiver noise figure. With the preamplifier added, the gain behind the QSD may be decreased by the same amount as that added on the front end. This will not only improve the NF of the radio, but will also reduce local oscillator spur amplification.

The RFE also includes an experimental impulse generator that will allow for computation of the QSD and sound card impulse response. The impulse response will then be used to dynamically equalize the I and Q signals in order to maximize image rejection.

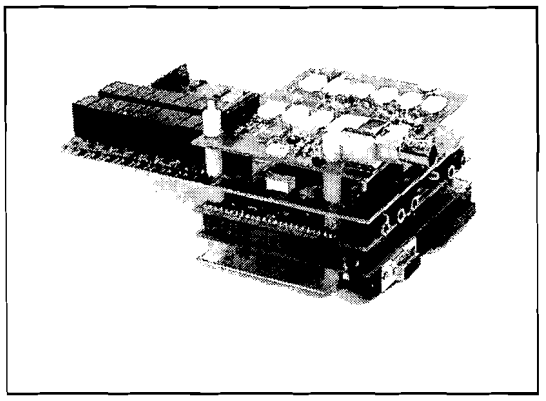

Table 1 provides dynamic range measurements that were performed on the SDR-1000 with RFE and using an M-Audio Audiophile USB in 16-bit mode. Two HP 8640B signal generators were combined through 16dB pads with a Mini-Circuits ZFSC-2-1-12 combiner. Port to port isolation was measured to be 80dB with a HP 8568A spectrum analyzer. Four different dynamic range options are possible using the 10dB attenuator and instrumentation amplifier (INA) gain settings. Third order dynamic range (or SFDR) is in the 88-90 dB range for all combinations. The noise figure is only 3dB for the entire receiver in the highest gain setting. Typically, the lower gain settings are preferable for all bands below 12M. All measurements are performed in a 500Hz bandwidth.

Gain Setting	NF	MDS	SFDR	Full Scale - MDS
0dB ATT, 26dB INA				
	3			
		-141		
			90	
				95
10dB ATT, 26dB INA				
	14			
		-130		
			89	
				94
0dB ATT, 0dB INA				
	17			
		-127		
			88	
				107
10dB ATT, 0dB INA				
	26			
		-118		
			89	
				108

Table 1 – Dynamic Range Experiments

The RFE sandwiches between the existing BPF and TRX boards so that the BPF provides front end filtering for the low noise preamp. The 1W PEP driver amplifier (OPA2674) will move to the RFE board as well so that the existing BPF board will filter its output. The RFE will also provide control signals for the 2M microwave transverter IF, 100W linear amplifier and automatic antenna tuning unit described below.

Provision has been made in the enclosure design to incorporate a Down East Microwave DEMI144-28ECK low-power transverter kit as seen in Figure 7. It is designed to function as an IF for microwave transverters. In receive; it uses a high-level double-balanced mixer (+17dBm) and a three chamber helical filter. It provides 50-100mW of linear output in transmit mode. TR control and RF interface is provided through a single coax connection to the RFE board.

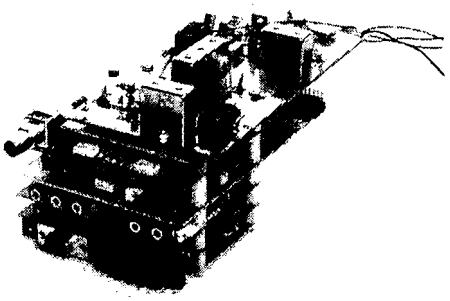

Figure 7 – DEMI144-28ECK 2M Transverter IF

The SDR-1000 has the provision to add an external oscillator for the AD9854 DDS. The unit ships with a 200MHz, low-jitter oscillator. Weak signal and microwave operation often dictates precise frequency control, including GPS locked references. The SDR-1000 can easily be converted to a 10MHz reference by cabling a 10MHz source to the oscillator connector, moving two pin jumpers, and setting the DDS PLL multiplier.

A 100W PEP linear amplifier with low pass filters is being designed to fit in the SDR-1000 enclosure. The TR relay and filter relays will be controlled from the RFE board. Further details are not available at the time of this writing.

To round out the SDR-1000 accessories, a third party automatic antenna tuner will be integrated into the packaging. Once again, control will be provided from the RFE board.

SDR Console Software – Continued Growth

One of the nicest parts of this entire project has been the tremendous outpouring of software, ideas, new concepts, and contributed software by many different individuals. Each software author's contribution is prominently listed in the source of the software which is released GPL. There are now at least three functioning console packages in addition to that offered by Flex-Radio and more are on the way. Home brewing of add-ons to radios is alive and well in this project and embodied primarily in the software.

Noise Blanking and Pulse Removal

One of the banes of narrow band receivers are the effects of pulse noise. The worst offenders are typically semi-periodic pulse trains such as line or alternator noise; but we would also like to reduce the impact of single large pulses like switch openings and closings. One day, while listening to the broadcast band and a horrible set of pulses arriving in a pulse train, we attempted a truly simple noise blanker. If a signal value rose too far above the Root Means Squared (RMS) value, it was blanked, or set to zero. The effects were immediate and surprising. We then analyzed many noise blanking circuits and found they did little more than this, but in many cases, required a pulse train to work properly. This pulse removal usually operates on the wider signal inside the "roofing filter" while the pulse is still narrow. Then when the blanked pulse sample is passed through the filters that follow, the filter acts like an interpolator to smooth over the hole you have made in the signal.

In the lab, a weak signal was dialed up on 40 meters. It was a South American station that was just above the noise floor. At the antenna, 4V pulses were added! The noise blanker was engaged and the weak signal rose out of the hash and was completely readable. It is clear we are able to not only duplicate the typical noise blanker, but also in some ways exceed its performance. But there is no reason to stop there. We can afford to do a more complex algorithm than this since we do not have to pay for the associated hardware. Our only cost is the time to code the algorithm.

There are two promising algorithms we are exploring. We have developed one such algorithm and it is now included in the SDRConsole. Image processing algorithms have often developed rank order statistics in an attempt to look in the neighborhood of a pixel to see if "it fit" into the overall picture. If it does not "fit in", then it is declared to be speckle noise. Its value in the image is replaced by a combination of the surrounding pixels that more fairly represents the area of the offending pixel. The technique works wonders in the removal of speckle noise from the image.

We wondered how well this might apply to the removal of pulse noise in one-dimensional signals such as ours. In fact, we found it had already been investigated. Sanjit Mitra of the University of California Santa Barbara wrote a paper in which he described this exact algorithm. In his paper, he explains how to calculate the statistics and performs several tests. He called it the Signal Dependent Rank Order Mean Noise Reduction algorithm. How it works is really straightforward. We will consider our digitally sampled signal in groups of five adjacent samples

$$X(t)=[x(t-2),x(t-2),x(t),x(t+1),x(t+2)]$$

We will take every sample but the middle one and sort them into an increasing value array.

$$W(t)=[w(0),w(1),w(2),w(3)]$$

We will compute from this ranked ordering, the rank order mean. This simply means we will take the middle two values and average them.

$$\mu(t) = [w(1)+w(2)]*0.5$$

We will compare the signal at time t, x(t), with this rank order mean and then to its rank ordered neighbors. We will set two thresholds. We will test to see if it departs from the behavior of its nearest neighbors at one threshold. We compare it again to its farthest neighbors in our four long rank order vector against a larger threshold. If it does depart from the behavior of its neighbors more than these threshold values, we will replace the signal with the rank order mean $\mu(t)$. This has an immediate impact on the processed signal versus the blanked signal. We do not just zero out the signal and pray the filtering which follows will fill in the hole adequately. Anyone who has listened to a receiver with an activated noise blanker adjacent to loud signals knows how the AGC and cut-off action of the noise blanker can be modulated by these strong signals. In the SDROM case, we replace the offending value with a value that is determined from a smoothing of the surrounding values.

Initial Test Results of SDROM

In support of the picture-is-worth-a-thousand-words argument, we include the following in Figure 8 from our Matlab experiments when developing the algorithm. We have four signals in this graph. The blue trace (top) is the incoming simulated signal. It was the test signal during our development. It is a two-tone signal plus noise. We have added pulses to the signal. We have zoomed into a region of 160 samples around a pulse. The red trace (second from top) is the raw signal without pulses. The green trace (third from top) is what a traditional noise blanker would do and the black trace (bottom) is the SDROM output.

The differences are subtle to the eye, but profound to the ear. The large spike is clearly evident. We chose this spike in order to more clearly demonstrate the differences in the algorithms because it occurred near a peak voltage in the signal. At sample 62, you will notice that the blanker has just zeroed the signal, which causes a sharp edge, ***and the attendant clicks*** (just like key clicks) that accompany such an occurrence. The final black trace, the SDROM pulse noise canceller, has replaced the pulse with a smoothed version of the signal without the pulse. There will be no key click-like phenomenon with this approach. A traditional noise blanker incorporates these sharp edges that spray energy all over the spectrum, just like a key click.

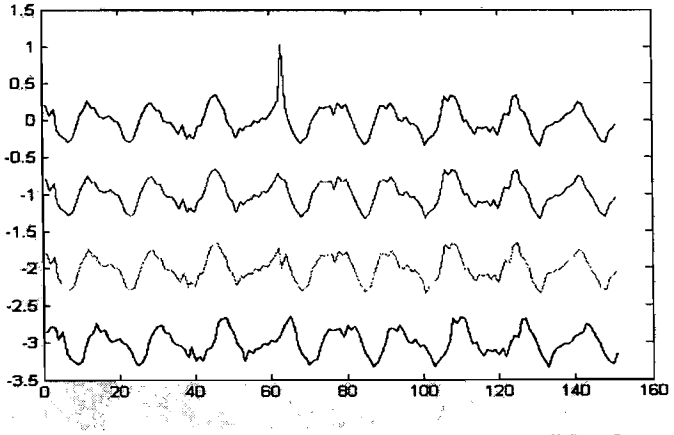

Figure 8 – SDROM compared to Noise Blanker

Nothing can more dramatically demonstrate this than the power spectrum.

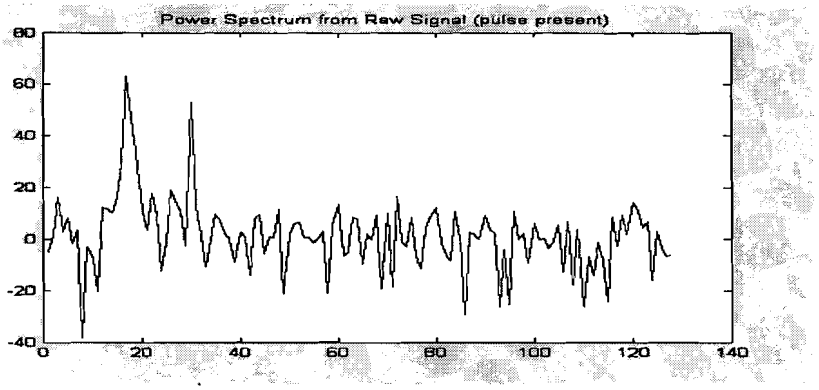

Figure 9 – Raw Signal

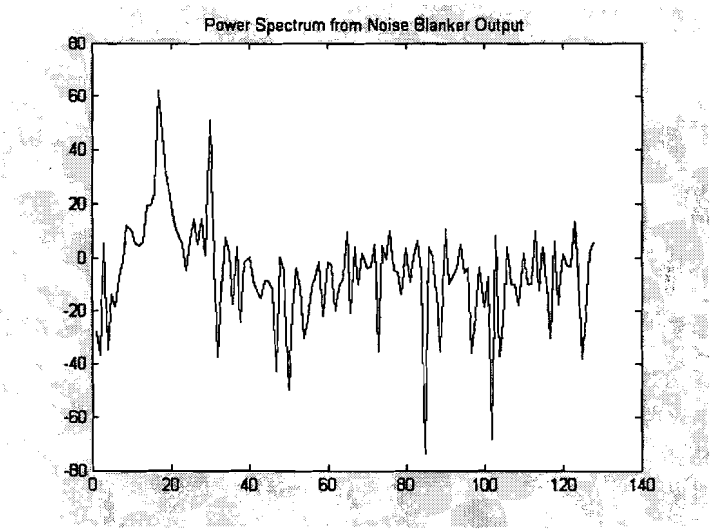

Figure 10 – Noise Blanker

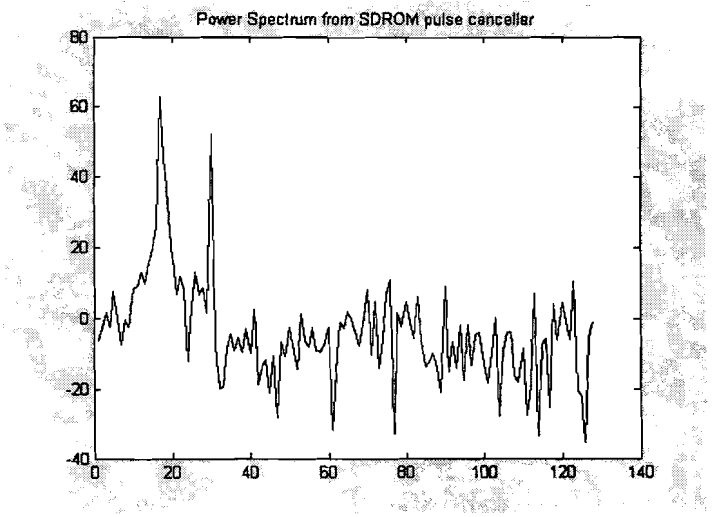

Figure 11 – SDROM

The differences are observed immediately if pointed out. The noise floor outside the area of the two tones is ***raised*** in the blanked signal over the signal with the pulse present. The SDROM output has the noise floor depressed by 10 dB from the signal plus pulse and more from the blanker. This extra energy will not be mixing in a nonlinear fashion with signals of interest in the SDROM output. It is a marked improvement over the noise blanker. But it is not perfect.

The primary drawback to the SDROM as implemented by Mitra, et. al. is that it treats the pulse in the same manner as the blanker insofar as it makes the assumption that a pulse is a single isolated event and limited to one sample. This is easily improved on by doing SDROM recursively. That is, we consider the filtered signal when deciding whether to replace a value. This will allow for wider pulses than spikes. We have not implemented this but we expect small dB improvement in the noise floor from this added wrinkle.

Improving Image Rejection

The ultimate in noise pulse removal would be to know what the pulse looks like and to subtract it from the signal. This might seem miraculous until you know that Leif Asbrink has approached this in Linrad as describe in several recent *QEX* articles. Leif attempts to find the pulses and subtract their pulse shape, using two parameters from the incoming narrowband signal: the phase angle and the amplitude. In our attempts to make a real radio out of the SDRConsole, it was thought to be a deficiency of this approach that it required the user to determine a single pulse shape through a measurement procedure done once and assumed correct. It is clear that this is not perfect, though it produces good results in VHF+ noise and has been utilized by many VHF/EME/Microwave users. The imperfection shows up in the need for Linrad to continue to blank with the zeroing algorithm those small pulses that make it past this. To the extent that you do not have the pulses correct, or the amplitude or phase correct, you are ADDING PULSE BACK into the signal. It seemed that we could, in fact, do a complete automatic job of determining more parameters and improving the performance.

It has been decided to add a single pulse-generating engine just before the mixer in the new add-on RFE board. Rather than reiterate all of the advantages of this new board, we will detail our approach to the pulse shape here. (See Hardware Enhancements section for more on the RFE board)

A terrific job of image rejection can be done if you know the phase and amplitude imbalance between the I and Q channel in the incoming signal. It would be ideal if this could be measured directly. We believe we can do this with a pulse generating mechanism. In our case, we need not know the exact relationship of our impulse response through our system, but rather its relative deviation from ideal. This is then easily added (convolved with) the filtering done for SSB, CW, etc. to remove the image in order to make both the I and Q response flat and equal with linear group delay across the spectrum of interest. In addition to accomplishing image rejection, this will give us the pulse shape of the ideal pulse entering the system and allow us to do a more complete job than Linrad can do with the one-size-fits-all impulse response. When we change bands (at a minimum), we will re-estimate the impulse response correction. Experimentation will allow us to determine if it needs to be done more often than once per Mhz change in frequency.

To that end, we derived a slow repeated pulse from a signal generator so that we could isolate one pulse at a time. The following graphs show a pulse as it has passed through the SDR-1000 hardware and has been captured upon passing through the most important element in any system, the sound card. The graph in Figure 12 show's the imaginary channel of the filtered signal in the area of one of these captured pulses. In a perfect world, this and its accompanying real part would be exactly the impulse response of the filter we have applied in the SDR-1000 console software that is applied for SSB detection. It would have nearly flat response in the passband and no phase or amplitude distortions. However, we live in the real world of real components of our mixer, instrumentation amplifiers in the SDR-1000 and op-amps in the A/D's in our sound cards. All contribute to distortion that hurts image rejection and keeps us from doing fancy noise reduction. Since we designed the bandpass filter in the IF, we know its impulse response perfectly. We register the location of the pulse and place it where we believe we have captured most of the response that can be seen above the noise floor. We compare that to the complex impulse response of the filter and correct for distortions. This will yield "perfect" image rejection. It will also allow us to apply the subtractive pulse canceller in a completely automated way since we will have to account only for perturbations from the ideal of the now determined impulse response.

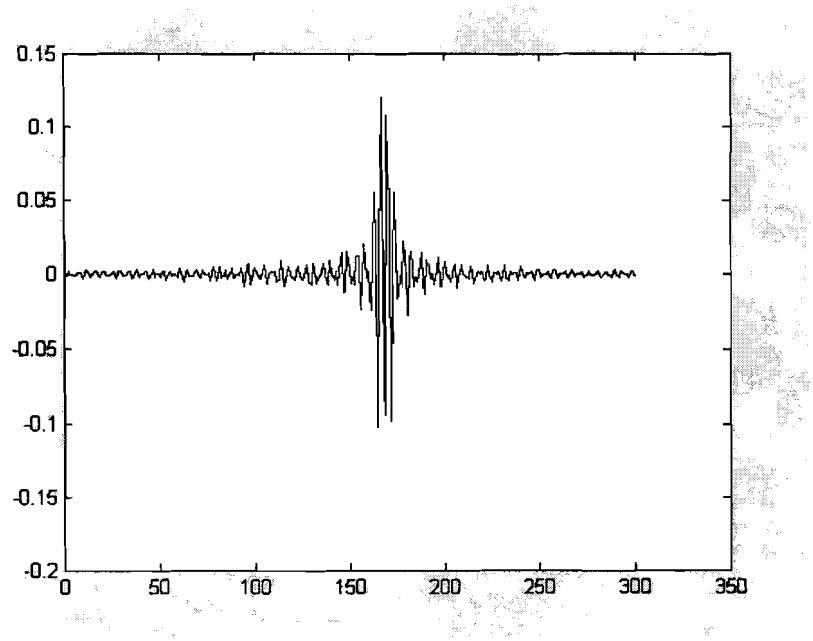

Figure 12 - A pulse time waveform in the imaginary channel

In most of the modern transceivers available to us, we have DSP processors which do automatic noise reduction and automatic notch filtering. Most of them (if not all) use a Least Mean Square (LMS) adaptive filter first described by Widrow, et. al. and commonly referred to as the "Widrow" filter. This filter has a serious drawback. It uses the longer term correlations present in tones, speech, or Morse to produce a filter which either enhances them and reduces noise or notches them if they are undesired. This correlation is done very weakly and at one lag or "look into the past". A steepest descent based on this stochastic

gradient is done. It is clear that this is a lossy and noisy look at the correlations needed to make this filter; yet it does work.

We would like to improve on this algorithm. One could use many different lags. As many lags as you will allow filter taps. This leads to a very expensive algorithm called Recursive Least Squares (RLS) and the unstable but fast versions of it known as Fast RLS or FRLS. Fortunately, there is another way that has recently been discovered outside of the area of noise canceling and notch filtering. We have adapted it for our use. It was developed in the echo-canceling world for multiple sensor microphone systems. Once you look past this function, you quickly see it is immediately applicable to other issues. It is known in that world as the Affine Projection Algorithm (APA) when you allow several more than one lag, but all the lags are adjacent to each other. There is an obvious extension of this to multiple lags that are not necessarily adjacent to each other, but cover a longer span. This can be extremely helpful in capturing more information about the signal. This version is known as Normalized Least Mean Square with Orthogonal Correction Factors (NLMS-OCF). A Google search will reveal numerous online documents if you need more detailed information.

Here, we will describe our first experiments and implementation. We have limited ourselves to APA in the Visual Basic console. This limitation will be removed in the upcoming versions of the console that will use other signal processing libraries and languages. For now, let's describe the results. We chose to use 3 lags to compare to our current signal sample in order to determine a good filter for our single experiment in notching a two-tone signal. We used the APA algorithm with a delay of 65 samples at 44100 samples per second. At that delay, we look at a filter with only 32 taps. We attempted to strain the algorithm with a short filter. In the end, we were amazed. Even in a noisy signal, given a short filter, we converged with the APA at a rate that was heretofore only achievable with RLS. In addition, it is automatically normalized for changing signal strengths due to AGC. It exhibits the better tracking behavior that is more akin to LMS, rather than RLS, which shuts down and stops listening unless you force it to listen by adding a memory leak constant to the RLS algorithm.

In 48 samples, we converged to a notch. In the power spectrum shown in Figure 13, we have two traces. One is before the automatic notch while the other is after. We have artificially shifted the notched spectrum down so they do not lie on top of each other. This extremely impressive result can only get better and more stable as we allow non-adjacent correlations of NLMS-OCF and improved performance across the spectrum if we choose distances that represent non-periodic signals more accurately.

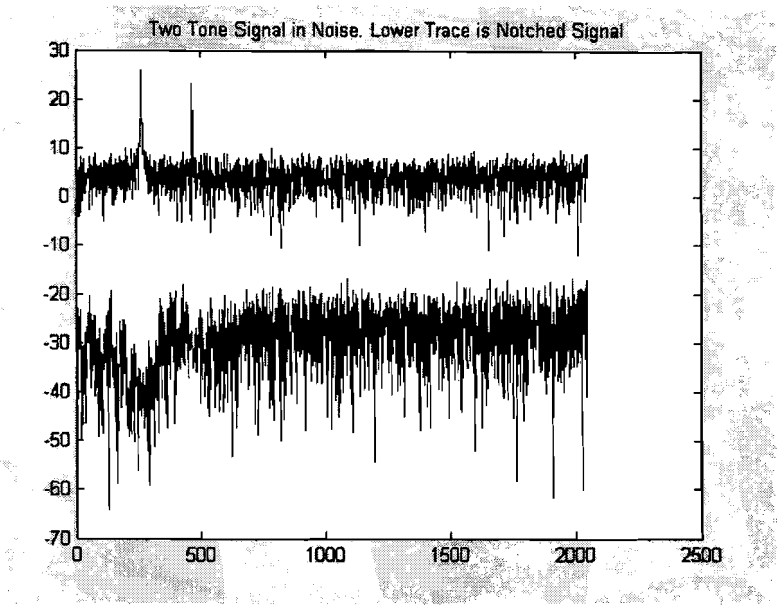

Figure 13 – Notched Power Spectrum

A serious drawback to the radio has been the way it operates as a CW rig, despite a Herculean effort by W5SXD. Frank Brickle, AB2KT, and author N4HY have come up with a technique to use Modulated CW (MCW) in order to gain full QSK without the drawbacks of using MCW. Since this is a software radio, we can afford to have an external circuit with a keyer chip and a tone generator that generates MCW. Some may be worried about spectral purity with this kind of excitation. However, we do not need to worry. We will do detection on the MCW signal inside the radio software rather than automatically transmitting it. We will then reconstitute the CW, with adjustable shaping and weight to be transmitted to the antenna. We will soon have RTTY and PSK31among the other modes already implemented. Rob Heard, in his own Delphi Version of the console, has implemented Slow Scan Television (SSTV) reception and is working on transmit capabilities. The full spectrum of narrow band communications on the ham bands is possible with this radio.

Recently, N4HY proposed an relatively easy scheme using the SDR to do frequency hopped spread spectrum using a compression-in-time algorithm that will enable the addition of synchrony signaling on every dwell while not losing or covering up the signal of interest.

Frank Brickle, AB2KT, has written an interesting article in a recent issue of *QEX,* which will lead inevitably to Cognitively Defined Radio. What this means is that the radio will detect a signal, classify it, and configure itself given the built in artificial intelligence to do so in the software defined radio.

AB2KT and N4HY are writing a full-blown console using Qt-Free as the GUI development engine and will be releasing the Linux/Alsa Sound/Qt console soon.

The following is a list of software developers listed in the latest beta version of the console software. The things they have contributed are too numerous to list but the radio would not be nearly as feature rich nor would it function as well without their contributions:

W5SXD, G6UVS, AA6YQ, W3IP, VE7APU, VK6APH, WK0J, N7TQM, N4HY, and AC5OG

New Object Oriented Architecture

While a firm groundwork has been established using the Visual Basic 6 (VB6) interface, it has become increasingly important to look at a new platform. With Microsoft making moves toward no longer supporting VB6[3], we began to look at building the SDRConsole in a more recent language. Rather than simply porting the current code, we decided to take a bold step and redesign the entire console from scratch.

Using the lessons learned from the VB6 design process, we began with a very high level view of the software and broke each section into smaller logical blocks. Examples of such blocks are Digital Signal Processing (DSP), DataStream and Hardware. These blocks would be further broken down to a size that is easily maintained. Dissecting the project in this way allowed us to break up the coding responsibility much more easily. Using a Unified Modeling Language (UML) tool called ArgoUML helped us to visualize our software model. Figure 14 shows a portion of an early prototype of this model. Given the open-source nature of our project, easing the ability for customers to contribute directly toward the development of this new platform would be crucial to the software's continued success. With open source code and customers contributing code in their own specific areas of expertise, we have a uniquely diverse development team.

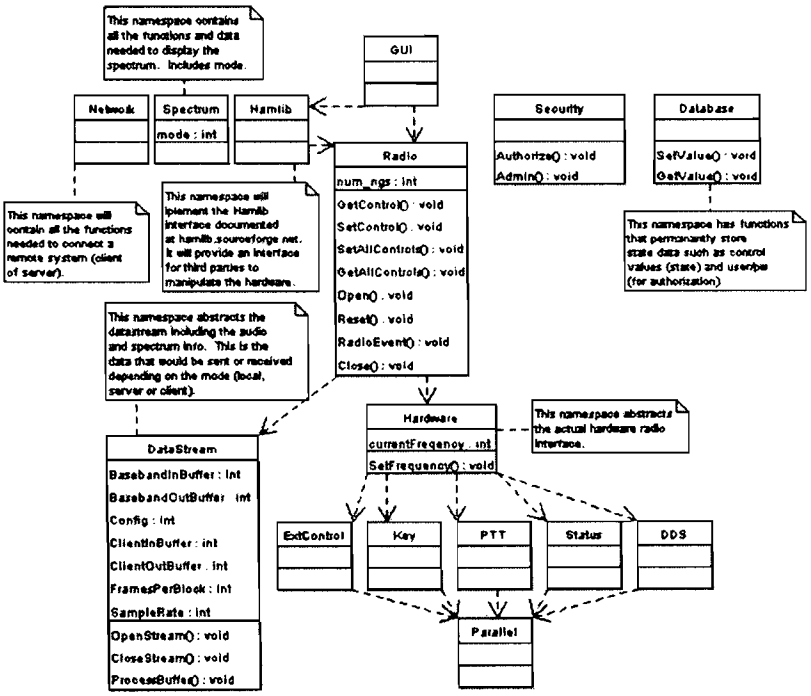

Figure 14 – UML Model

One of the major design points in our new software model was to provide support for both a binary executable interface as well as a web interface that could be accessed from any computer with access to the Internet. This would not be an easy undertaking, as we would have to consider such things as audio compression, data encryption and serialization. Despite the development cost, this feature is necessary for a cutting edge product such as the SDR-1000 and would open new doors of opportunity for remote radio applications. Imagine being able to pull up your radio interface and even transmitting using your PDA. This is the type of flexibility that we are aiming for with our new design.

In our search for an appropriate language, our options were somewhat narrowed by our strict criteria. While still wanting to offer an easy-access version for beginning programmers, our product would shine best in a multithreaded environment. We essentially needed something with the easy visual interface that Visual Basic offered while at the same time offering the power of C or C++. After a bit of research, C# seemed to be the logical choice. Further investigation revealed an extensive class library in the .NET Framework. With defined namespaces such as System.XML, System.Threading and System.Security, we would be able to quickly integrate powerful features such as RSA encryption, multithreading and serialization without spending months developing these libraries on our own[4]. Linux and C#.NET versions of the console are being developed by Flex Radio, N4HY, and AB2KT. We expect these versions to be in Beta testing within the next month.

One of the more exciting possible applications with the new object oriented architecture and the use of more modern development tools available in C#.NET and Linux is the easy ability to remote the transceiver hardware and to do the signal processing at the other end of the remote connection. An extremely exciting prospect for doing coherent combining of the signals from multiple radios is immediately available to the serious experimenter with minor modifications to the radio to allow for the injection of coherent DDS oscillator signals.

Having discussed the web-based access to the radio, it becomes obvious that not everyone is equally endowed with Internet connections. It would be very easy for the local server running, for example, a RealServer to save a known user's configuration, internet connection speed, and other factors so that when the unique user connects to the radio, the remote transmission to that user is configured appropriately for them. This information is easily stored in a database local to the radio and accessed upon connection.

In the early days when we were doing the initial development on the radio, some obvious expediencies came immediately to bear on the issue of getting the radio finished and software developed that would enable the experimenter to begin tinkering. AC5OG, as the developer, was not a DSP expert and not a real-time computing expert. A decision was made to use Intel's Signal Processing Library (SPL). This enabled fast, accurate, well written algorithms to be immediately available to the Visual Basic 6 console without having to be written from scratch, debugged, and optimized. However, as we move on to do other things with the radio and as Intel has dropped its support for SPL and substituted a fee for license based library known as PPL, we have decided to explore other options and some have become available for our experimentation with the new object oriented console.

We would like to re-use code across all platforms whether it be Linux, MacOS, or Windows. Recent developments have helped tremendously in that regard. A project that solves most of the really tough issues of dealing with audio and sound card issues was found in the PortAudio API. The PortAudio API project is on the web at http://www.portaudio.com. It has versions that enable one API to be used on Linux, MacOS, and Microsoft Windows. All versions of the code can have one interface to the sound system in the computer on which they are running. Eric Wachsmann has written a C# wrapper to talk to PortAudio for Microsoft Windows Visual Studio .NET 2003 and the API already comes native to run on Linux, and Unix (including FreeBSD which will run on the Mac).

We need a similar kind of library to do the primitive signal processing procedures that SPL did for us. In addition, we wish to begin doing the APA, NLMS-OCF, N-channel combining algorithm work and experiments not yet conceived by us but which we are fully aware will have features in common. The primary features will include a solid library to do linear algebra and matrix/vector manipulations as well as optimized fast Fourier transforms (FFT).

The latter was available for Visual Studio 6 as well as Linux and the Unices in the form of FFTW (see http://www.fftw.org). N4HY has recently ported FFTW-2.1.5, the most recent release version, to Microsoft Visual Studio (MSVS) .NET 2003 with all the project and solution files. This is available through a link on the Flex-Radio web site on the resources page.

The signal processing and linear algebra routines have been captured in a U.S. government supported open source effort known as VSIPL (see http://www.vsipl.org). Heretofore, no one known to us had ported this library to Microsoft Windows tools. It compiled and ran natively on Linux, Unix and MacOS (FreeBSD) systems. N4HY has managed to get all versions of VSIPL, which is written in C, to compile and make static and .dll libraries for MSVS.NET 2003. All library versions including using FFTW-2.1.5 as well as the native FFT in VSIPL, both static and dynamic have been made and tested.
This library is also available as a link on the Flex-Radio resources page.

---

[1] G. Youngblood, AC5OG, "A Software Defined Radio for the Masses: Part 1," *QEX*, Jul/Aug 2002, pp. 13-21; "A Software Defined Radio for the Masses: Part 2," *QEX*, Sep/Oct 2002, pp. 10-18; "A Software Defined Radio for the Masses: Part 3," *QEX*, Nov/Dec 2002, pp. 27-36; "A Software Defined Radio for the Masses: Part 4," *QEX*, Mar/Apr 2003, pp. 20-31.

[2] Marvin E. Frerking, *Digital Signal Processing in Communication Systems*, Kluwer Achedemic Publishers, Norwell, MA.

[3] Product Family Life-Cycle Guidelines for Visual Basic 6.0, http://msdn.microsoft.com/vbasic/support/vb6.aspx, Accessed 24 Feb 2004.

[4] Class Library, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/cpref_start.asp, Accessed 24 Feb 2004.