Linrad: *New Possibilities for Communications Experimenters, Part 3*

Linux *and the* Linrad *software package.*

By Leif Åsbrink, SM5BSZ

Linrad is a computer program that runs on PC computers under the *Linux* operating system. I have chosen to write this software under *Linux* because under *Linux*, all development tools are free. I want to encourage others to play with the code and make additions and modifications. The disadvantage is that one has to install *Linux*, which is seen as a major obstacle by many potential users. Installing *Linux* is quite simple on a reasonably modern standard PC. It is possible to run *Linrad* on a fast 486 computer, but it is not trivial to install *Linux* on such old machines. The easy way to install *Linux* is to use a computer that can be booted from a CD. Just insert a *Linux* distribution CD, boot from it and follow the instructions—not really difficult, but if you have never done it before, it helps a lot to have someone more experienced at your side.

To run *Linrad*, you must install *svgalib*, a package that contains drive routines for video boards. Unfortunately, *svgalib* does not support all board types; if the video board is integrated on your motherboard, there is substantial risk that *svgalib* will not support it. The sound system under *Linux* is not quite stable yet. Some *Linux* distributions have sound that works automatically, but most need the installation of a new sound package. I am using OSS, which is not free; but the free ALSA sound system should be compatible, although I have not been able to make it run myself.

Your sound package must support the sound card in your computer, which is often a problem with sound cards integrated on the motherboard. I am not an expert on hardware compatibility issues. Everything I have tried has worked automatically but I have had several reports about problems with integrated video or sound. This article does not further concern itself with *Linux* issues that are not specific to *Linrad*.

Basic Functions of *Linrad*

Linrad displays a portion of the RF spectrum on the computer screen. The bandwidth (kilohertz) displayed is determined by your hardware. The operator can see all signals within the on-screen passband and can tune to any particular station by clicking it with the mouse. The signal is then

Jaders Prastgard 3265
63505 Eskilstuna, Sweden
leif.asbrink@mbox300.swipnet.se

processed and presented to the operator via the headphones or as text on the screen. *Linrad* is planned to support the following modes:

- CW (including weak-signal and meteor-scatter)
- SSB
- FM
- Various digital modes

Linrad is still at a reasonably early development stage. Only the weak-signal CW routines are implemented and the only output is sound to the headphones. The weak-signal CW algorithms are currently used for the other modes, but they are not necessarily optimal. When the user has pressed the button, *Linrad* looks back in time and decides what the optimum center frequency is and how it changes with time. An internal local oscillator is set up and used to convert the frequency of the incoming spectrum for the desired signal to be centered in the final passband selected by the user. This AFC function makes it possible to use narrower filters than otherwise possible on weak and unstable CW signals. The output of the final filter can be used in several ways. It can be

- Routed directly to the headphones
- Coherently processed (only the component of the signal that is in phase with the carrier is sent to the loudspeaker)
- Filtered through a narrower filter for one ear or delayed.

All tricks I know of to enhance the reception of weak signals are there.

For normal CW or SSB, one can disable the AFC, set the bandwidth to 2 kHz and use *Linrad* as an ordinary receiver. There is no AGC, however, since an AGC is inappropriate for weak-signal CW. When the algorithms for other modes are in place, they will have AGC and some interference-fighting functions.

Linrad can combine signals from two antennas. The spectrum presented on the screen will then contain all the signals from both antennas and when the mouse is clicked, the two signals are combined to maximize the desired signal. This is an obvious strategy for weak-signal CW, where the two antennas are typically the two polarizations of a crossed Yagi array. The effect of combining the signals is identical to that obtained when the crossed Yagi array is aligned with the polarization of the incoming wave. The user is then presented with the best possible signal. This function is very useful for EME. It eliminates loss of signal caused by Faraday rotation. For other modes, it may be better to combine the two signals for minimum interference, which can be done by hand (but which will be automated in the future).

Combining two antennas with complete freedom in phases and amplitudes means electronic lobe control. Someday, when the antenna signals can be digitized more or less directly at low cost, many more channels will be extremely useful on crowded HF bands, but *Linrad* has no provisions for more than two channels yet.

Linrad contains a very special noise blanker, such that *Linrad* can be used to hear signals that are impossible to receive with any other radio. Power-line noise may have repetition rates of several kilohertz. The interference source is a spark gap in series with a capacitor, that is, a defective insulator in series with some good ones. When the mains frequency is applied at a high voltage, pulse trains are emitted at twice the mains frequency. Each pulse train typically consists of 10 pulses. A noise blanker must resolve the individual pulses well and therefore the noise blanker must work with a bandwidth of 10 kHz or more. Many ham transceivers have such blankers and they work fine until some strong undesired signal is present within the blanker bandwidth. The *Linrad* blanker has automatic notch filters that remove all strong signals in the blanker passband. The *Linrad* blanker finds out as much as it can from both of the receive channels and subtracts the most probable interference waveform from the incoming data. That means that much less information is lost because only a few data points at the peak of the pulse must be blanked. The pulse tails are accurately reduced to well below the noise floor by the subtraction process.

The practical implementation of *Linrad* is based on Fourier transforms. The description above is what the algorithms do, not the way they are implemented. The Fourier transforms are needed anyway for various reasons and it saves a lot of CPU time to use them for the actual processing.

Linrad has no prejudice about how you want to use it or to the hardware to which it is connected. The user must select parameters that make a good radio receiver from the building blocks. If you ask for a 20-Hz filter that rejects signals well only 0.1 Hz outside the passband, the processing delay will be on the order of 10 seconds for the filter. If a simple sine window is chosen to save CPU time and large decimation rates are selected, aliasing spurs will degrade the dynamic range, and so on. There is nothing wrong with any of this, and it has nothing to do with *Linrad* being a DSP system. The propagation delay through a filter is related to the *Q* regardless of whether an analog or a digital implementation is chosen. There may be perfectly valid reasons to design a radio either way, and *Linrad* allows you to do it. I have tried to make the control functions so that it is intuitive to set up a good, normal radio receiver.

Linrad Block Diagram

The block diagram of Fig 1 illustrates signal flow through the different processing steps of *Linrad*. Blocks labeled *fft* and a number go from the time domain to the frequency domain via a fast Fourier transform routine. Blocks labeled *timf* and a number go from the frequency domain to the time domain by an inverse-fast-Fourier transform routine. Filtering and resampling are extremely easy in the frequency domain. One just removes irrelevant parts of the spectrum to get rid of undesired frequencies. The smaller transform size automatically reduces the sampling rate of the time-domain function that follows.

There are no conventional DSP filters inside *Linrad*. The filter shapes are not controlled by FIR filter coefficients. The actual processing is the same, however; the same sums of products are taken, but in another order. The window function used for the FFT becomes one component in the effective filter function that is used to prevent aliasing spurs in the resampling transformation.

The theory may seem frightening; but when actually operating *Linrad*, it is not difficult to understand what is going on. For example, you may try a very bad (and very fast) FFT with no window and few points. The resampling spurs that then surround any strong signal are easily seen. It is not difficult to figure out what causes them and how to remove them. It is exactly as in any other radio. If the frequency conversion reduces the frequency by a large factor, one has to put more effort into the filters to avoid spurs. The same thing happens when reducing the sampling speed. A large reduction rate requires a narrower filter because resampling spurs come closer.

The processing is controlled by certain parameters. These parameters fall into four categories:

- Hardware related parameters.
- Receive-mode related parameters.
- Operator's current preferences for a particular station.
- Dynamic parameters calculated by *Linrad* itself.

The hardware parameters are set for a particular hardware combination as a part of the *Linrad* installation.

The receive-mode related parameters enable or disable certain functions and control how much memory *Linrad* will allocate for various purposes. They also allow experimentation with transform sizes and other things one usually would not like to change when operating in a particular mode. What the parameters are and how to set them up is described in conjunction with detailed descriptions of the different processing blocks below. The operator's current preferences are given to *Linrad* by mouse clicks on the screen. Some parameters like the angle and phase that control how the two antenna signals are combined can be either category 3 or 4. A parameter of category 3 "Adapt" or "Fixed" is used to select that. Most of the real-time parameters are obvious. There is a help function; pressing the F1 key gives information about where the controls are and how to use them. The F1 key can also be used to get information about the mode related parameters in the mode-setup routine.

How to Install *Linrad*

First, you need a computer with *Linux* installed on it. You then must install *svgalib-1.4.3* or later if it is not already included in your *Linux* installation. You also must install *nasm*, an assembler that reads Intel-style assembly language exactly as written without making all sorts of unpredictable (to the innocent beginner) assumptions. *Linrad* contains fast routines that use the Intel multimedia instructions MMX and XMM and they are written in assembly language. Finally, you must install *Linrad* itself.

Linrad, *nasm* and *svgalib* are all available on the Internet. The *Linrad* home page is at **antennspecialisten.se/~sm5bsz/*Linux*dsp/*Linrad*.htm**, with mirrors at **www.g7rau.co.uk/sm5bsz/*Linux*dsp/*Linrad*.htm** and **nitehawk.com/sm5bsz/*Linux*dsp/*Linrad*.htm**. You can find links to *svgalib* and *nasm* there. At the *Linrad* home page, you will also find links with descriptions for the novice about how to install the *svgalib*, *nasm* and *Linrad* packages. *Linrad* comes only as source code, so you must compile and link it to obtain an executable program. Everything is automated. You need only issue two commands: **configure** followed by **make**. This is a normal procedure for the installation of a *Linux* package.

Before starting *Linrad*, you must get sound going. Installing OSS, the sound package from 4Front Technologies is always easy if your *Linux* distribution is modern enough to still be supported. Some distributions have sound included that works directly with standard sound cards. Sound under *Linux* is still a bit messy, but presumably, ALSA will be a free and well-working part of *Linux* in the near future if it is not already.

My information about *Linux* sound has a link from the *Linrad* home page, but my information is getting old. Since I have paid the OSS license, I just go on using OSS without much concern about what happens to *Linux* sound in general. There is a *Linrad* mailing list where users can interchange information. You will find a link to it at the *Linrad* home page.

Setting Computer-Related Parameters

When starting *Linrad* for the very first time, the initial screen in text mode may contain warning messages about multimedia instructions that you will never see again unless you remove the parameter file *par_userint*. Your hardware may support multimedia instructions while your *Linux* kernel does not. *Linrad* allows you to use multimedia instructions even if the system flag says they are illegal. This works fine under *Mandrake* 8.1. I do not know if it is because *Linrad* is the only software using the multimedia registers, so it therefore does not matter whether these registers are handled properly on task switching by the kernel.

From the initial screen, you can go

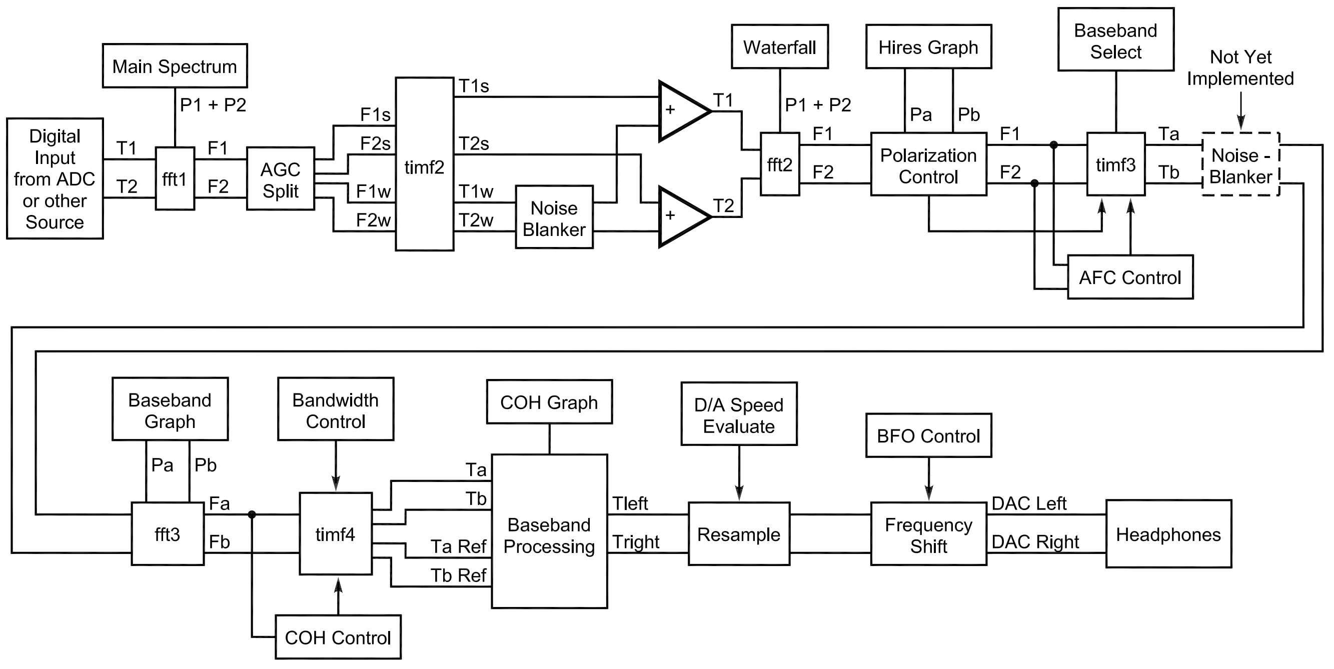

Fig 1—The block diagram of *Linrad* with two receive channels and the second FFT. T1 and T2 are signals in the time domain from two antennas 1 and 2. F1 and F2 are the corresponding signals in the frequency domain. Ta and Tb are linear combinations of T1 and T2 that make the desired signal zero in Tb and consequently maximizes the desired signal in Ta. TaRef is a time function constructed from a much narrower bandwidth than Ta. For Morse coded signals, it will be the CW carrier that is useful for coherent processing.

only to the video-mode selection where one of the graphic modes reported by *svgalib* can be selected. *Linrad* needs 256 colors and a minimum screen width of 640 pixels. I recommend a 1024×768 screen. If you do not see the video modes expected for your hardware, something is wrong with *svgalib*. It is possible to change the device drivers included in *svgalib* by changing its *Makefile.cfg* file and then recompiling.

After selecting screen size, you need to select a font size. Start with the minimum size. Finally, you are prompted for mouse speed, where 64 is a suitable starting value. If your mouse type is not the one presented on the screen, you need to edit the */etc/vga/libvga.config* file. After you hit ENTER, *Linrad* will switch to the graphic screen you have selected. If the screen looks okay, save the settings you've made so far by pressing *W*. The screen you should have at this point is the main menu. In case the screen does not look okay, it is possible to instruct *svgalib* to change the video signals by editing the */etc/vga/libvga.config* file. You may also try another screen size.

Setting the SoundCard Parameters

Linrad assumes you always use the same hardware to feed the computer with digital data. Therefore, the sampling speed and data format is set only once, from the main menu. Press U to set the soundcard parameters. If you try something else, you will be prompted to this setup anyway if parameters are undefined.

Linrad opens all device drivers it can find. Some of them may be defective: They may belong to some other sound system than the one you actually have running. There may be several soundcards in the computer. The user should know which drivers belong to which soundcard, but you may try what seems reasonable. On a simple, standard system where there are no alternatives, *Linrad* selects the only possible alternative automatically.

After having selected the input device, you need to select the sampling speed and the input format. Do not set a higher speed than necessary. Oversampling will load the CPU, but it will not improve performance much. When everything is set up, you may check that the sampling speed is correct for your hardware by checking how a strong signal is attenuated as you tune it upward in frequency. When it reaches the Nyquist frequency (half the sampling frequency), it is folded

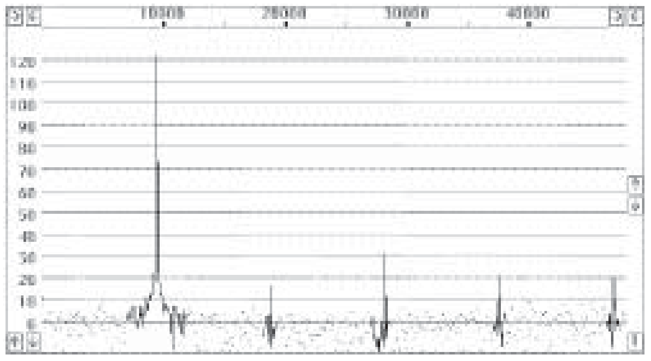
Fig 2—Unwindowed Fourier transform in 512 points. The bin bandwidth is 93.75 Hz and the near saturating signal occupies a single bin only because the frequency is carefully adjusted to be exactly on the center of an fft bin. This graph like Figs 3 to 8 are *Linrad* screen dumps on which straight lines are drawn between pixels for better visibility.

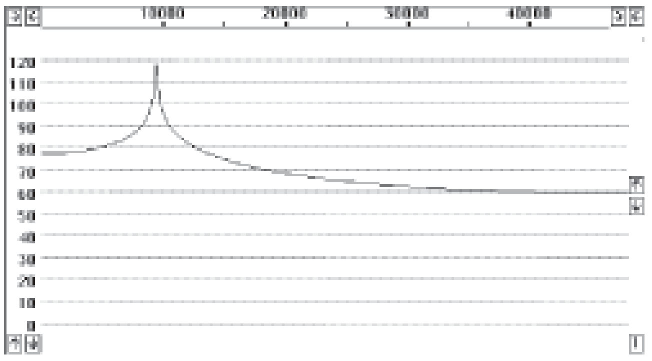
Fig 3—Same as Fig 2, but this time the frequency is adjusted to a point right between two frequency bins. Unwindowed ffts provide poor dynamic range and a poor shape of the spectrum peaks.

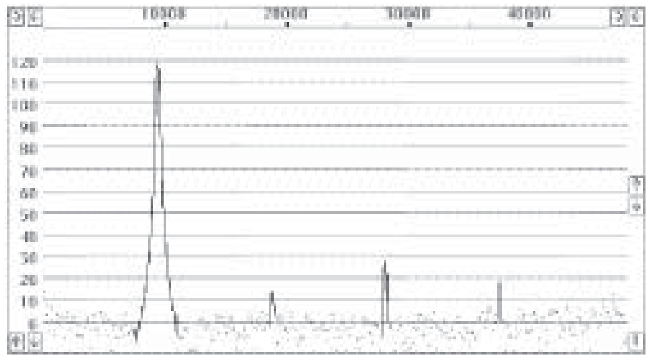
Fig 4—A sine^4 window provides a nice spectrum with a peak shape that is nearly independent of how the frequency is related to the fft bins. The peak is about 2.5 bins wide at the 6-dB points. This is because the window makes so many data points very small so the length of time during which the window does not attenuate very much is only 40% of the total transform time.

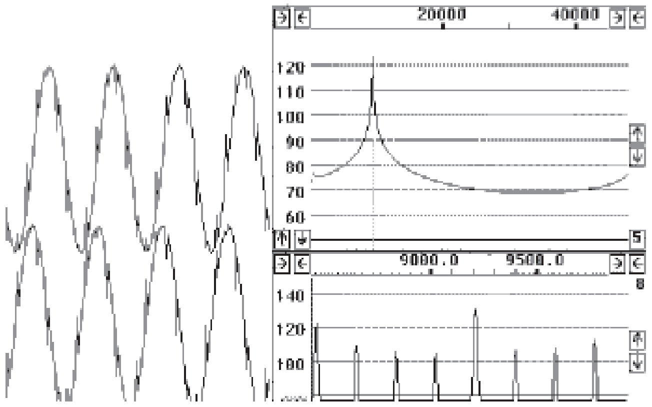
Fig 5—Unwindowed back transformation. Eight data points from the unwindowed transform are used to compute the baseband signal by back transformation. The time function comes in blocks of eight points, and there is a discontinuity each time two time functions are joined together. In this figure, the eight points are centered on the strong signal. The baseband spectrum computed from this time function shows these discontinuities as spurs, which are only about 20 dB below the signal. The origin of the transients is explained in the text.

back as an alias signal that appears to go down in frequency while the real signal is still tuned upward. The sampling speed should be set high enough to make the alias signal attenuated well enough when it has reached into the desired passband. The *Linrad* setup dialog is intended to be reasonably self-explanatory, and the *Linrad* homepage has links to detailed information.

Parameters That Depend on Receive Mode

The First Group: fft1

There are several distinctly different modes of operation for *Linrad*. As of this writing, only the weak-signal CW mode is partly implemented. It is possible to set weak-signal-CW parameters to get a system that is reasonably well adapted for SSB or normal CW. These modes can be selected from the main menu so you can switch quickly between modes, but the code actually executed remains that for weak-signal CW, until dedicated routines for the other modes are in place.

The total number of receive-mode-dependent parameters is quite long. The mode-dependent parameters are set from several screens; you are prompted to them when running a new mode for the first time, but you can also go there from a menu. Each screen controls its own part of the block diagram shown in Fig 1. By simply pressing ENTER at each of the parameter screens, you will select default parameters that will usually provide a reasonable starting point.

The first processing block has these parameters:

- First FFT bandwidth
- First FFT window (power of sine)
- First forward FFT version
- First FFT storage time (s)
- First FFT amplitude
- Enable second FFT

These parameters control block *fft1* in Fig 1. The bandwidth and the window control the size of the FFT. It is in powers of two, so you do not get exactly the bandwidth specified by the parameter. The window controls the shape of the filter associated with each FFT bin. A higher value yields a wider bandwidth in FFT bins but steeper skirts; a larger FFT may be required to get the desired bandwidth in hertz. A higher value also causes more CPU processing load because transforms must overlap more. The different FFT implementations may run at different relative speeds depending on processor and memory architecture. The storage time is important only if the second FFT is disabled. It tells *Linrad* how much memory to allocate for old transforms; they are used for AFC and spur rejection in the absence of the second FFT. The amplitude parameter can be used to fool *Linrad*. This, in case the signal level is too high, causing the noise floor to be treated as a strong signal—something likely to happen if *Linrad* is used to process an ordinary .WAV file.

When the second FFT is disabled, the output of *fft1* is routed directly to *timf3*, disabling the noise blanker. The input to *fft1* is the raw data from the sound card or raw data from a file. The input may be one or two channels in real or complex format as defined by the sound-card parameters or the raw data file. In the block diagram of Fig 1, T1 and T2 represent the two time functions from two antenna signals. The help key, F1, can be used to get some information about the effects of these parameters.

When choosing bandwidth and window, you build the basic filter used in the *fft1* block. Figs 2 and 3 show what happens when the window parameter is set to zero (no window). These figures were produced by running *Linrad* with a direct-conversion receiver at 2.5 MHz while sending a very pure sine wave at 2.509 MHz into it. In Fig 2, the frequency is adjusted for the signal to be located exactly on one frequency bin while Fig 3 shows what happens when the signal is located right between two bins. The figures are unaveraged power spectra of size 512.

The great difference between Figs 2 and 3 results because the Fourier transform is the spectrum of a signal that is obtained when the 512 samples of the input data are repeated continuously. When the frequency matches a bin exactly, the first point will fit exactly to the sine wave when placed as point 513 for the next repetition. When the frequency is exactly halfway, point 1 will not fit at all when placed as number 513. There will be a large discontinuity, and that discontinuity has signal energy over the entire frequency spectrum.

Unwindowed FFTs are very fast, but they do not allow a large dynamic range. When a group of bins is picked to make a rectangular filter, the stop-band attenuation will only be about 50 dB for a 512-point FFT. With larger transforms, the situation improves because the discontinuities repeat less often; but the phenomenon is still there.

The way to cure the problem is to use a window. Fig 4 shows exactly the same signal as Figs 2 and 3 but here a sine^4 window is applied. With such a high-order window, it does not matter whether the signal is centered on a bin or not. The points toward the ends of the time sequence are all extremely small, so the discontinuity is very well suppressed. Notice that the maximum is much wider. There are three points on the peak now, so a 2048-point FFT will be necessary for the same inherent bandwidth.

When *Linrad* is used as a radio receiver, a window power of two or three is sufficient. When using it as a high-performance spectrum analyzer with the first FFT generating broadband spectra, however, windows up to a power of nine may be used. That makes it possible to study extremely low levels of unintended sidebands on the test signals, if the hardware has the quality to allow it. The hardware used to produce Figs 2-4 represents the unit described at **antennspecialisten.se/~sm5bsz/*Linux*dsp/rxiq/opt2500.htm**. The unit can be purchased from **www.antennspecialisten.se**.

The choice of bandwidth and filter depends on what you are going to do with the output data. When the second FFT is disabled, the output is displayed *and* is used to produce a frequency-shifted signal in the time domain by *timf3*. When the decimation rate for *timf3* is low, the alias spurs from the *timf3* point-decimation process are far from the desired frequency. Then the basic filter skirts need not be very steep to suppress alias signals well. When the decimation rate is high, the filter requirements are higher. High decimation rates allow the computer to do advanced processing on many signals simultaneously. When the hardware is an ordinary SSB receiver, the dynamic range within the SSB bandwidth would likely be below 50 dB anyway, and then it does not matter much what window you choose.

Bandwidth selection depends on a compromise between the resolution you desire on the screen and the time delay you are willing to accept. If you want 1-Hz resolution with a sine^4 window, each transform spans about four seconds and a corresponding time delay is unavoidable. A high resolution is required to make the AFC follow extremely weak and unstable CW signals well enough to use coherent processing. AFC is not required for stable signals, and then there is no need for high resolution. A bandwidth of 20 Hz gives good visibility for CW signals on the waterfall graph with a modest processing delay. By setting a large bandwidth for *fft1*, it is possible to make very fast waterfall graphs from which you can read high-speed CW signals directly.

When the second FFT is enabled,

the output of *fft1* is used to produce notch filters for all strong signals in the passband. There is no reason to select a narrow bandwidth, but good skirt steepness is required. Typically, 200 Hz and sine[3] will fit high-performance broadband hardware. If you are going to select a very narrow bandwidth for the second FFT, it may be a good idea to select a smaller bandwidth for *fft1*. The ratio of *fft1* to *fft2* has some influence on how deep the notches for strong signals must be. If your computer is capable of making the *fft2* bandwidth 0.2 Hz, it would be unlikely to have problems making the *fft1* bandwidth 10 Hz or so, particularly since a less-demanding window would then be required.

Linrad has oscilloscope functions that allow you to see the time functions. With a signal generator, you may check the spurious responses that occur as a result of inadequate filters both in the time domain and in the frequency domain. With a poor filter, you will easily see the resampling spurs with the second FFT disabled.

The upper right part of Fig 5 shows the *fft1* spectrum in 256 points from 0 to 48 kHz of the same signal as used for Figs 2-4. No window is used.

The spectrum is frequency shifted to place the desired signal at zero frequency. The baseband signal is then filtered out by multiplying the transform with a nicely rounded filter function. For Figs 5-8, this filter is in eight points. The baseband signal is then obtained from an inverse FFT. The filter makes the 248 points outside the baseband equal to zero, so there is no reason to do the inverse transform in more than eight points. This is how decimation comes in, naturally. The two sine waves at the left are the resulting complex-time function. This is the baseband signal I and Q for the selected frequency. These sine waves are made up from segments of eight points, each spanning the same time as one frame of *fft1*. Since the number of points is reduced by a factor of 32, the bandwidth is reduced from 48 kHz to 1.5 kHz with a sampling speed of 1.5 kHz. The bandwidth is actually reduced a bit more because of the rounded shape of the filter used to pick the eight points. The lower-right part shows the baseband spectrum in 256 points. The signal itself is at 9200 Hz and the resampling spurs are separated by 187.5 Hz, the inverted value of the time for one *fft1* frame. The accuracy of this spectrum is poor towards the ends, the screen shows the spectrum divided by the filter function, now expanded from eight to 256 points and this division becomes division by zero at the ends.

The resampling spurs show up in the time function at the left side of Fig 5 as spikes that repeat at an interval of eight points. The explanation of why these spikes occur is that an unwindowed FFT has to use all of the frequencies to reproduce a sine wave in case it is not exactly at one of the bin frequencies. That is what we see in the *fft1* spectrum, the signal is at about 70 dB or more at all frequencies. When only eight points are selected, the inverse transformation will not give back the original signal; it gives something that differs at the beginning and end of the time interval. Successive transforms do not fit well to each other so spikes are generated. The way to avoid this problem is to use interleaved transforms and not use the end regions for the inverse transform. The associated spectra are then calculated with windowed transforms.

Fig 6 is the same as Fig 5 except that here, the 1.5-kHz segment used for the inverse transform is centered 3 kHz above the strong signal, which is now outside the range used for the inverse transform.

As can be seen in the main spectrum, the level of the unwindowed transform is at about 85 dB at 12 kHz and that signal, which is an artifact, shows up as spurious signals separated by 187.5 Hz.

What we see in Fig 5 and Fig 6 is what happens when resampling is done without an appropriate anti-alias filter. As was pointed out, with appropriate references, by Gerald Youngblood, AC5OG, (*QEX*, Nov/Dec 2002, p 31), filtering in the time domain and filtering in the frequency domain are equivalent. The problem is that the filter used to generate Figs 5 and 6 was not the nicely rounded filter in eight points. It is the sum of the output from eight filters (with frequency responses like the unwindowed FFT) summed together with weights according to the nicely rounded function.

Fig 6 clearly shows what is required to suppress the alias spurs. By using only the four center points of each transform and calculating twice as many interleaved transforms, we can construct a baseband time function from the inverse transform that does not include the end discontinuities. *Linrad* does it by applying a window because the power spectrum is needed for AFC, spur cancellation and to produce graphs. The transforms are interleaved for the 3-dB points of successive window functions to match the same points in the time function. To avoid problems related to the AGC (the gain may change between two transforms), *Linrad* makes soft

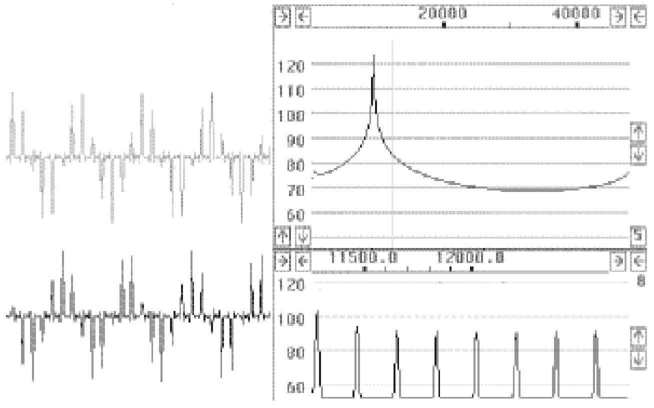

Fig 6—Unwindowed back transformation. Same as Fig 5 but this time the eight points used for the baseband are centered 3 kHz above the strong signal. The signal itself, the sine and cosine is now outside the baseband, but the discontinuities are very strong. As can be seen from the main spectrum, they are present at all frequencies.

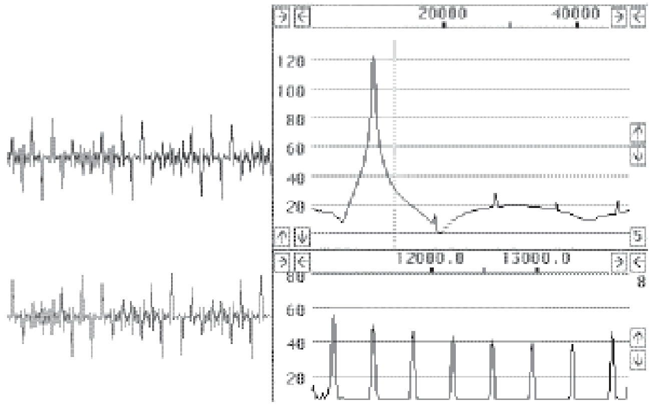

Fig 7—A sine^2 window makes the spectrum look much better. With the high dynamic range of the hardware used to produce these images, the sine^2 window is not good enough to bring the discontinuities below the noise floor with a small transform size like 256.

transitions between successive inverse transforms. A sine-squared window produces the filter function shown in Fig 7. When the baseband is filtered out by weighting together eight such filters, the alias signals are reasonably well suppressed as can be seen in Fig 7, both in the baseband time function and its spectrum.

By selecting a longer *fft1* transform, it is possible to suppress the alias spurs below any desired level. This is done by setting the "First FFT bandwidth" parameter narrower or by setting a narrower window function by making the "First FFT window (power of sine)" parameter larger. Fig 8 shows what happens if a sine^4 window is chosen for a size-256 *fft1*.

It is self-evident that one can avoid spurs completely by use of a transform size and window shape that reduces contributions from undesired signals to well below the noise floor for all data points used in the inverse transform. When a large baseband bandwidth is desired, the offending signal may be included among the data points used for the inverse transformation. This will be perfectly okay if all data points containing energy from the undesired signal are picked with exactly the same amplitude. As already mentioned above, the frequency range to be back-transformed is selected with a nicely rounded filter, which may lead to unequal amplitudes for the different data points on a strong signal. By selecting the bin bandwidth of *fft1* much narrower than the baseband bandwidth, one can make the difference in amplitude small, thereby placing these spurs below the noise floor.

When the second FFT is enabled, all the frequency bins of *fft1* are inverse transformed. Frequencies where strong signals are present are attenuated, however, and that creates a similar problem to the one just described. If the window or *fft1* size is inadequate, spurs will show up in the waterfall graph. By injecting a near-saturating signal into the antenna input, one can verify that such spurs are invisible and well below the noise floor of the hardware in use.

The Second Group: AGC, timf2 and fft2

These blocks are present only when the second FFT is enabled. They are controlled by the following parameters:

- First backward FFT version
- First backward FFT attenuation *N*
- Second FFT bandwidth factor in powers of two
- Second FFT window (power of sine)
- Second forward FFT version
- Second forward FFT attenuation *N*
- Second FFT storage time

When the second FFT is enabled, the main purpose of the first FFT is to find the frequencies on which strong local signals are present. Such frequencies are attenuated by the AGC to the extent that the strong signals will not overflow when further processing is done with 16-bit arithmetic. The AGC block splits the output of *fft1* into two groups. One contains the narrow-bandwidth signals that have been located in *fft1*, F1s and F2s in Fig 1. The other essentially contains the noise floor from the rest of the spectrum, F1w and F2w in Fig 1. The two sets of *fft1* transforms are back transformed in the *timf2* block to produce two sets of time functions; T1s and T2s contain strong signals, while T1w and T2w contain weak signals.

The noise blanker operates on T1w and T2w only. That means that it operates on a signal that has passed notch filters removing all strong signals that otherwise would have disturbed the operation of the blanker. The *Linrad* noise blanker is far more sophisticated than a conventional noise blanker and will be described separately.

Once noise has been removed from the weak signals, weak and strong signals are added together and sent to the *fft2* block. That block again converts the signals to the frequency domain with the output signals F1 and F2 as shown in Fig 1.

Since the AGC ensures that the data will fit into 16 bits, selecting the first inverse FFT implementation that uses MMX multimedia instructions saves a lot of CPU time because MMX is about three times faster than floating point. Likewise, the second FFT should be set to use MMX if the processor supports it.

Since some processing is designed to use 16-bit numbers, it is essential to make sure that the signal levels are set properly. The "First backward FFT attenuation N" and "Second forward FFT attenuation N" parameters set the signal levels by telling how many of the FFT butterfly loops should contain a right shift. Each right shift divides the signal level by two, which is attenuation by 6 dB. Gain set too high will cause saturation, while gain set too low will degrade the noise floor by the addition of quantization noise. For details, follow the link "set digital signal levels correctly" on the *Linrad* home page.

The parameter "Second FFT storage time" tells how much memory to allocate for old frequency-domain data. If the computer has enough memory, it is advantageous to allow a long time here. The AFC will be limited to the time interval specified here, and if the computer has plenty of memory, there is no reason to not use it.

The Third Group: AFC and Spur Cancellation

When this group is enabled, a group of parameters defining maximum AFC lock range, maximum number of spurs to cancel and so forth, must be selected. These parameters essentially allocate memory and set the upper limit for the operator's current preferences that he can select with the mouse. The AFC will lock to the strongest signal within the specified range, and to avoid locking to stronger undesired signals close to the desired one you may need to use a narrow lock range.

The functions in this group use the frequency-domain data from the entire spectrum. The input is the output of *fft2* if enabled, otherwise the input is the output of *fft1*. The third group will also contain the automatic Morse-code-to-ASCII translation, but this part is not implemented yet.

The Fourth Group: Baseband Processing

This is the last group of mode-

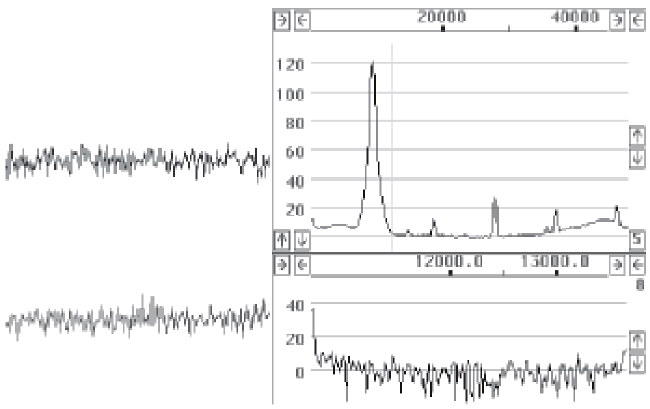

Fig 8—A sine^4 window is enough even on a small transform like 256 to make the near saturating signal narrow enough not to enter the baseband even though the baseband starts only 1.5 kHz above the strong signal.

dependent parameters. They are:

- First-mixer bandwidth reduction in powers of two
- First mixer number of channels
- Baseband storage time
- Output delay margin
- Output sampling speed
- Default output mode
- Audio expander exponent

The first-mixer bandwidth reduction is simply the decimation rate in the frequency-shifted inverse transformation that produces the baseband signal as discussed in conjunction with Figs 5-8. By selecting a large decimation rate, one saves CPU time and memory, but the baseband bandwidth is reduced at the same time.

The baseband noise blanker between *timf3* and *fft3* should operate at a bandwidth that is well above the bandwidth of the desired signal. As long as it is not implemented, there is no reason to use baseband bandwidths that are more than two times larger than the bandwidth of the desired signal. The factor of two is needed to accommodate a filter function.

The way the baseband signal is extracted from the broadband input signal is very efficient, and it is possible to process a large number of signals simultaneously. At present there is no reason to select more than one channel because only one channel can be sent to the loudspeaker/headphones. More channels will be useful when the Morse-code-to-ASCII routines are in place. For the other parameters, press F1 with the mouse cursor on one of them to get information.

The Receive Screen

Once the mode dependent parameters are set, *Linrad* enters receive mode and presents the receive screen to the user. The following windows are present:

- Main spectrum and waterfall
- High resolution spectrum
- Baseband spectrum
- AFC window
- Polarization control
- Coherence graph
- EME window
- Frequency control

Buttons in each window allow the user to change processing parameters. The spectra can be zoomed in and out and the user may move the windows around and set the sizes of the different spectra as desired.

The basic operation of *Linrad* is extremely simple. Move the mouse cursor onto a signal that is visible in the main spectrum and click the button. The corresponding signal, filtered through the baseband filter, will immediately be sent to the loudspeaker. The shape of the baseband filter can be modified with the mouse and the baseband frequency is shifted to the desired audio frequency by setting the BFO frequency.

All the processes are more or less automatic; but to take full advantage of them, one needs a basic understanding of how they work. Subsequent articles will describe some of these processes in detail. Some information can be obtained with the F1 help key, and there is a lot of information available at the *Linrad* home page.

Summary

This article has concerned itself with the coarse structure of *Linrad* and its block diagram. The significance of window functions for dynamic range has been illustrated; this problem is always encountered in SDRs when a signal is resampled at a lower speed. Subsequent articles will focus on how to use the special features of *Linrad* to improve receiver performance. ☐☐