8 Software Defined Radio

Phil Harman, VK6APH and Steve Ireland, VK6VZ

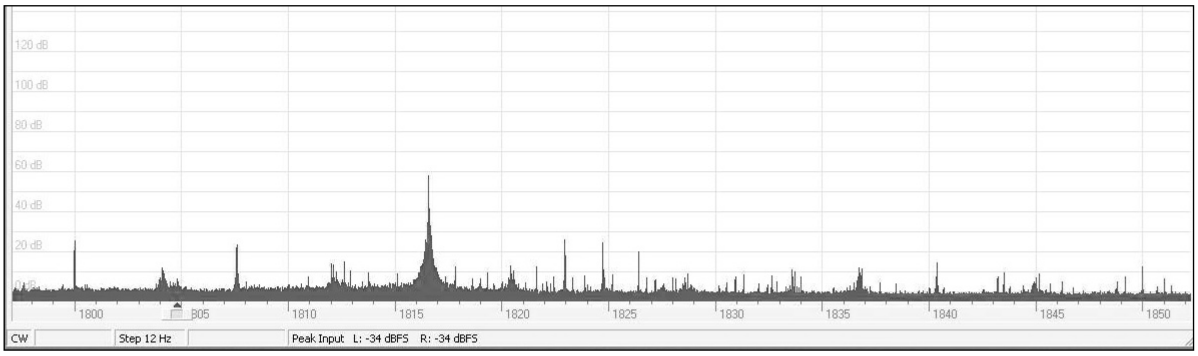


Fig 8.1: Part of an SDR-eye view of the 1.8MHz band

Those who have read amateur radio journals and magazines closely over the last few years will have seen a large number of references to software defined radios or 'SDRs'. If you are a traditionalist who likes his radios to fit onto the tabletop, have big knobs, meters and dials that glow, the idea of a radio made of software isn't going to be exactly appealing. However, it is in terms of old-fashioned radio values - such as sounding great and dealing well with weak signals in strong noise - that SDRs hold perhaps one of their two biggest advantages over conventional superheterodyne radios.

However, perhaps *the* biggest advantage is that SDR software lets you *see* radio signals - not just one at a time as you would hear them on a conventional radio, but all those that are present in a reasonable chunk of an amateur band. This is possible by means of highly-sensitive bandscopes which display signals down to the nano-volt level. **Fig. 8.1** shows a view provided by the bandscope of the free *Rocky* SDR software by Alex Shovkoplyas, VE3NEA [1]. It illustrates the 1.8MHz band on a PC screen, when coupled to SoftRock SDR receiver hardware [2].

Not only are we talking about 'seeing' radio signals but this is done in 3D - signals can be seen in their amplitude, frequency and time dimensions. Very expensive conventional superheterodyne radios can also do this now, but arguably not as well as SDRs. Those who have seen the built-in bandscopes that are now standard on top-of-the line amateur radio transceivers costing several thousands of dollars will notice that, for example, Rocky offers a much bigger and more dynamic view of a piece of radio spectrum than the former.

Tuning for weak DX signals can be a pretty hit-and-miss affair, as Murphy's Law of DXing dictates that as soon as you are about to tune across that rare station that is calling CQ, he will cease calling. In contrast, with a SDR bandscope, rather than tuning your radio, with the right analogue-to-digital converter you can see all the stations in a 192kHz (or possibly larger) window and click your mouse (or similar) onto whichever station you desire to listen to. The bandscopes in both *Rocky* and the *KGKSDR* [3] software by Duncan Munroe, M0KGK, (see **Fig 8.2**) can be adjusted in size to covers the entire width of your PC screen!

In 2006, VK6VZ first tested a SoftRock 40 hardware/*Rocky* software SDR on weak CW DX signals on the 1.8MHz band against a Yaesu FT-1000 transceiver. He found that of every four signals he could see on the SDR, he could usually only find one of them by careful tuning of the FT-1000 (unless he cheated and peeked at the SDR screen). Tuning between four signals using *Rocky* and the SoftRock hardware was simply three clicks of the mouse. As a CW operator for 37 years with over 220 countries confirmed on 1.8MHz, VK6VZ is no slouch at tuning to find weak signals, but reckons his country count would be much higher if he had been using an SDR for all this time.

In addition to a bandscope, which has a high resolution in frequency, most SDRs now offer a display which has a high resolution in time, known as a waterfall display (see **Fig 8.3**), owing to

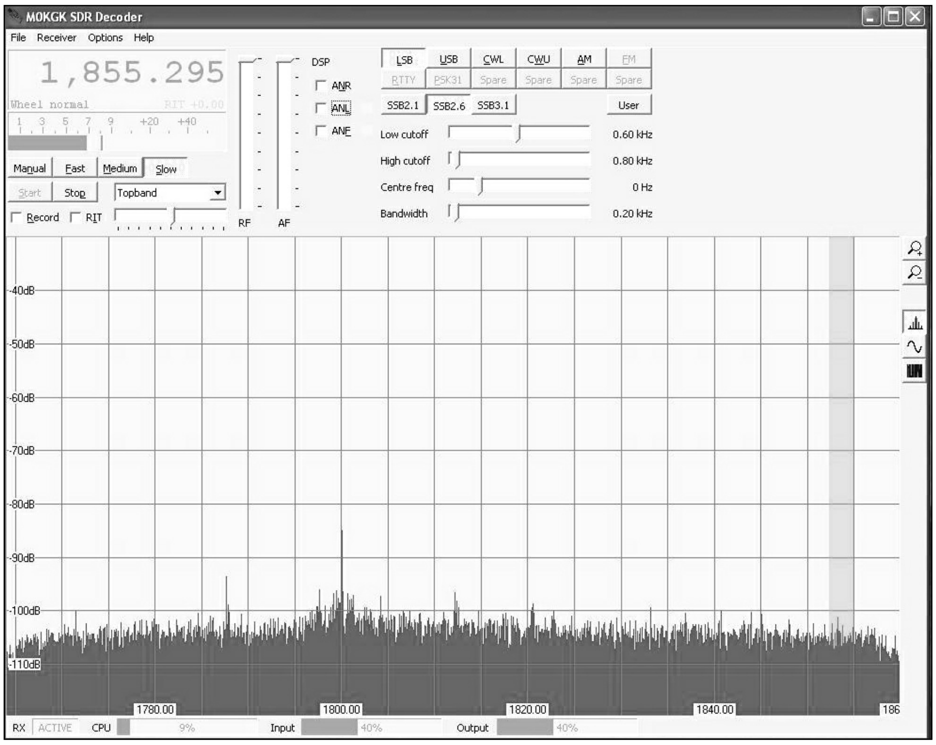


Fig 8.2: Bandscope using Duncan Munroe, M0KGK's *KGKSDR* software

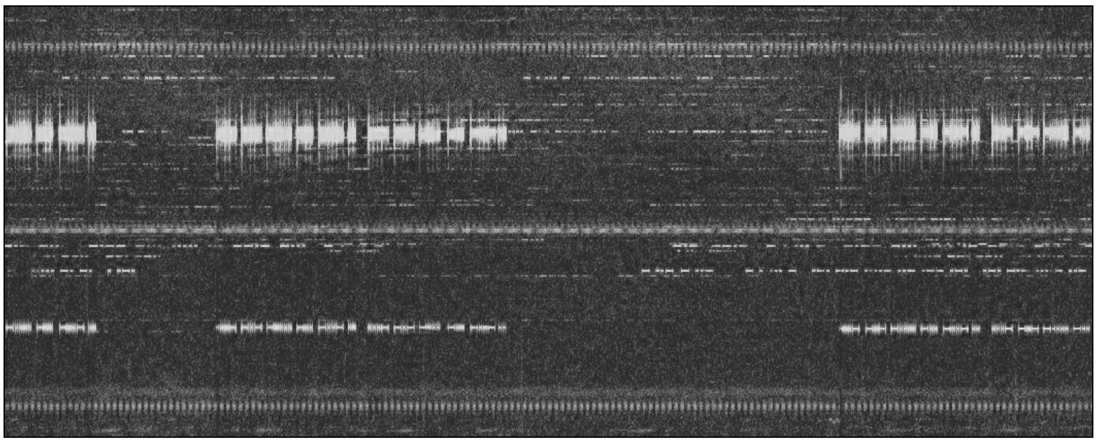


Fig 8.3: A waterfall display

the way it 'flows' across the screen from left-to-right or top-to-bottom. The waterfall display on *Rocky* even enables high-speed CW signals - up to about 40 words per minute - to be visually copied. Waterfall displays also show up flaws in the spectral purity of signals - in particular the key clicks generated by some modern transceivers. No more disputes as to how bad a friend's key clicks are - if you have *Rocky* or a similar program you can e-mail them a screen shot of their signal, captured from the waterfall display.

Over the last few years, some very significant developments have been made to SDRs, which has taken them out of the realm of the experimentally-minded and into the world of the hard-core radio operator. However, before exploring them, we need to look at some of the basic principles of SDR. Let's start with a definition for an SDR that both the authors like:

"A software defined radio refers to wireless communication in which the transmitter modulation is generated or defined by a computer - and the receiver uses a computer to recover the signal intelligence. To select the desired modulation type, the proper programs must be run by the computer which controls the transmitter and receiver."

It is in the recovery of the signal intelligence that the advantages of an SDR lie - in which a digital signal is converted into an analogue one, at which point essentially (almost) all the signal filtering and processing is carried out.

When VK6APH and VK6VZ first started writing about SDRs in *RadCom* magazine several years ago, essentially all the SDR hardware designs intended for amateur radio were based on a quadrature switching detector (QSD) followed by an analogue-to-digital converter (ADC) which used a PC sound card or chip (see **Fig 8.4**). The highly popular SoftRock series of simple receiver and transceiver kits and the Flex-Radio SDR-1000 (and its successor) the Flex-5000 [4] uses this design.

However, a second SDR hardware architecture of Digital Down Conversion (DDC)/Digital Up Conversion (DUC) is now emerging, where a high-speed ADC is connected directly to the receiver antenna - see **Fig 8.5**. Examples of receivers using the DDC technique are the Perseus [5], SDR-IQ [6], Hans Zahnd HB9CBU's ADAT ADT-200A [7], the High Performance Software Defined Radio (HPSDR) Mercury [8] and the Quicksilver QS1-R [9].

VK6APH was converted to SDR from analogue radios through the purchase of an early version of the Flex-Radio SDR-1000, while both VK6APH and VK6VZ have been greatly enthusiastic about the SoftRock receiver kits. VK6VZ has used various Softrock receiver versions extensively on 1.8MHz and as the basis for an out-board bandscope for an FT-1000 transceiver and, latterly, an Elecraft K3. However, time has moved on and, right now, both us are primarily interested in the use of DDC hardware-based receivers and companion DUC transmitters, configured as a transceiver.

The reason for this 'sea change' is that ADCs suitable for building direct sampling receivers , such as the Linear Technologies LT2208, are now available that provide sufficient bits-per-sample so as to give blocking dynamic ranges over 100dB - comparable to, or better than, some top-of-the-range conventional HF transceivers.

As an illustration of this high performance, the figures are given nearby for the final 'alpha' version of the open-source HPSDR Mercury DDC receiver, in which VK6APH has been involved in the design and the boards are now being produced by TAPR [10]. As you can see from **Table 8.1**, Mercury has a Blocking Dynamic Range of 119dB, which is basically independent of frequency spacing.

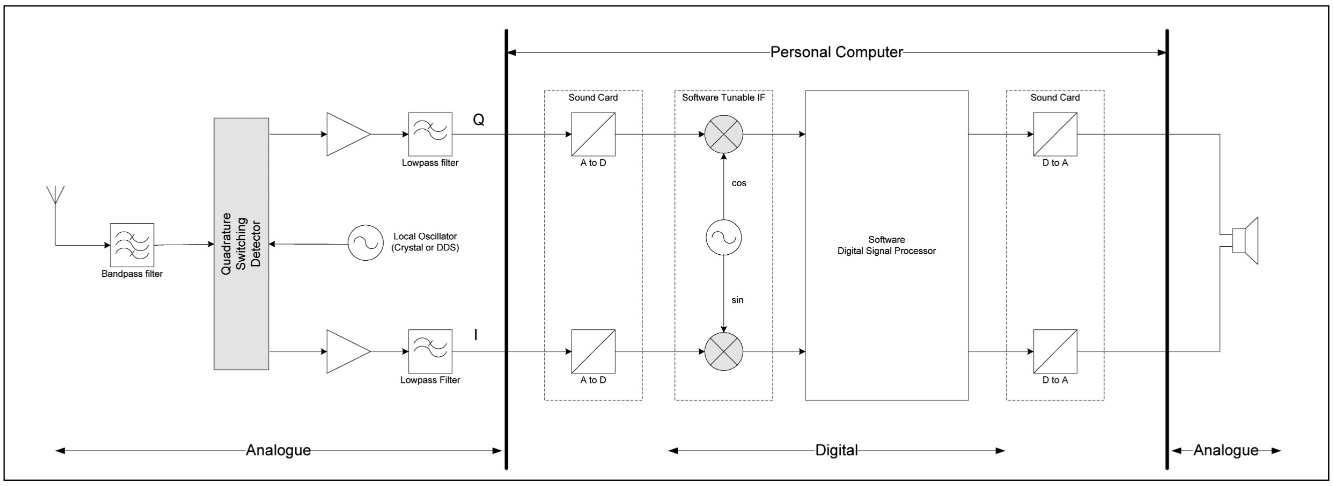


Fig 8.4: Block diagram of a basic SDR, using a quadrature switching detector (QSD)

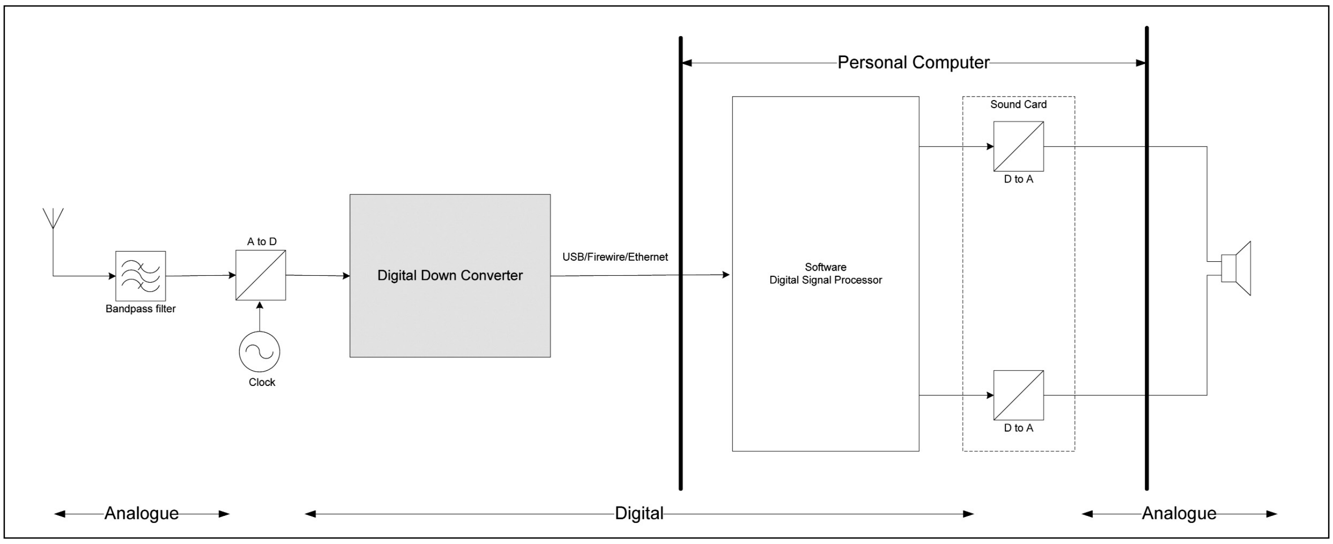


Fig 8.5: Block diagram of a basic SDR, using digital down conversion (DDC)

VK6APH has been using prototype HPSDR transceiver hardware for over a year now. VK6VZ has moved onto converting his DDC receiver hardware into DDC/DUC transceiver hardware similar to that used by VK6APH, with the addition of the HPSDR Penelope DUC exciter/transmitter board and a 100W PA from a surplus commercial HF transmitter. As we write in early 2009, a growing number of radio amateurs around the world are taking to the airwaves with 'home-made' transceivers, based on the Mercury, Penelope and the Ozy boards which are being sold by TAPR.

In the time that SDRs have moved towards DDC/DUC hardware architectures, the world of analogue transceivers has also moved on, with superb performance (assisted by digital signal processing and filtering) offered by radios such as the Ten Tec Orion, Icom IC-7800, Yaesu FT-2000 and, latterly, from the Elecraft K3. The Elecraft K3 currently tops Rob Sherwood NC0B's famous Receiver Test Data table [11] with a wide-spaced BDR of 140dB at 100kHz, but this falls to 101dB at 2kHz. (For more on commercial equipment, see Peter Hart's contribution at the end of the HF Transmitters and Transceivers chapter.) In contrast, although the HPSDR Mercury's BDR is about 21dB worse at 100kHz-wide signal spacing than the K3, the Mercury appears to be about 18dB better than the Elecraft K3 at a 2kHz spacing - which is obviously important if there is a very strong interfering signal near to the one you are listening to. On the other hand, if you have a very strong signal about 100kHz away from the signal you are listening to, then the K3 should cope with this better.

ADC overload	-12dBm (preamp on), +8dBm (preamp off)
MDS (500Hz) all amateur bands (1.8MHz to 50MHz)	-138dBm (preamp on), -118dBm (preamp off)
MDS (500Hz) on 50MHz via HPSDR Alexaires preamp	-146dBm
IP3 equivalent	+33dBm (preamp on), >50dBm (preamp off). Note the IP3 is independent of tone spacing.
Blocking Dynamic Range	119dB. Blocking Dynamic Range (BDR) was measured at 100kHz and 5kHz for 1dB gain compression with similar results.
122.88MHz clock phase noise	-149dBc/Hz at 1kHz spacing.

NOTE: The BDR is set by the overload point of the ADC rather than being phase-noise limited.

Table 8.1: HPSDR Mercury DDC receiver performance figures

In general terms, the BDR of the new breed of analogue radios is similar to that achieved in DDC SDR hardware and the former also offer bandscope facilities and easily-updatable firmware - two of the major advantages that SDRs have had almost to themselves in the past. The analogue radios also offer a very familiar user interface - knobs and buttons, rather than a mouse and a keyboard. So why would users actually want to use a true DDC/DUC SDR, even given its superior bandscope facilities, rather than an up-to-the-minute analogue superheterodyne-type radio with a really good DSP back-end?

The answer is very simple and comes down to one of the most crucial features of any radio receiver - because signals on a DDC SDR sound clearer and better. There is also another reason - for those of us that use computers every day, keyboards, mouses, windows and pull-down menus are now even more familiar and intuitive to use than knobs and buttons.

Owing to the signal processing and selectivity of SDRs being provided digitally by a computer rather than crystal filters, you can provide continuously variable selectivity down to a few tens of Hertz - properly designed digital filters don't ring like crystal ones. Similarly, noise filtering/blanking on SDRs are better than anything we have experienced on analogue radios - or using the digital signal processors that are available as add-ons or fitted to the current generation of HF transceivers.

Both VK6VZ and VK6APH have owned Yaesu FT-1000MPs - a fine transceiver with a relatively good built-in DSP noise reduction and filtering - but the DSP processing available in today's SDR software such as the free and open source *PowerSDR*^{TM} [12], written mainly by Bob McGwier, N4HY, and Frank Brickle, AB2KT, and VE3NEA's free *Rocky* is vastly superior to that available in the FT-1000MP in our opinion.

One of the best things about SDR digital filtering is that it is usually continuously variable. For example, by simply clicking with your mouse onto the filter bandwidth screen icon on the *Rocky* spectrum display and dragging it, you can vary the bandwidth of selected filter - in the case of the CW filter from 600Hz down to 20Hz. This means the operator can actually optimise the bandwidth of the received signal, in terms of signal-to-noise ratio. As VE3NEA notes on his website, for CW signals in white noise this is 1.5 times the words per minute of the CW signal you are listening to - which for a 30WPM signal is about 45Hz. VK6VZ uses this control on *Rocky* (which he uses in conjunction with SoftRock receiver hardware) by simply dragging the filter bandwidth narrow/wider until the weak CW DX signal he is listening to sounds most readable. On a noisy evening on 160m this often seems to be around 150Hz.

The second key advantage of SDRs is that they are much nicer to listen to in terms of their audio quality than conventional communications-type radios (more of this in the next section) which are based on the superheterodyne design. Most of the receiving process is digital, and hi-fidelity digital signal processing can therefore be applied. The combination of no crystal filters and a high-quality PC sound-card makes current SDRs sound, as one American acquaintance aptly observed: "as though you are directly connected to the ionosphere." Without loads of mixers and crystal filters in the way, good signals sound really good and bad signals sound terrible - whereas a modern multiple conversion superheterodyne receiver/transceiver makes them all sound pretty much as muddy/woolly as each other.

Back in October 2008, three radio amateurs got together in the VK6APH workshop - VK6APH himself, VK6VZ and our friend Fred, VK6GE (an Elecraft K2 owner, very long-time radio amateur and retired professional radio operator). On the bench were an Elecraft K3 and a HPSDR Mercury (using the open-source *PowerSDR*^{TM} software), both connected to VK6APH's Moxon Claw HF beam antenna via an antenna switch. In this case, the bandwidth of the HPSDR was not being varied continuously to improve signal-to-noise ratio, but 'standard' filter bandwidths (ie 2.4kHz and 500Hz) were being selected in *PowerSDR*^{TM} that matched the filters that were available in the K3.

For 45 minutes or so, we sought-out weak SSB and CW signals on a noisy 14MHz band and switched the antenna from one radio to another - just simple A/B testing. The verdict was unanimous - whilst the Elecraft K3 was probably the best-sounding (and performing) analogue radio the three of us had ever used, the HPSDR sounded 'better' and weak signals were considerably easier to understand.

Why was this so? The performance figures for the two radios are very similar! The answer is actually very simple and relates to the crystal filtering ('roofing filters' in the case of the K3) that the K3 - and most analogue communications radios - use.

All the radio frequencies that amateurs use are covered in noise - some of which is atmospheric, some of which is ionospheric and the rest is man/machine made. When noise pulses/spikes pass through a crystal filter, the phase response of the filter differs, depending on the noise frequency. However, when noise pulses/spikes pass through an ADC with a linear response, the phase response stays the same, because the ADC treats them in a linear manner.

That's the theoretical explanation - what happens in practice is that on a DDC SDR any noise actually sounds mellow and easy-on-the-ear, in a manner that has to be heard to be believed. In the case of an analogue radio, even one stage of crystal filtering (such as is used on the K3) is enough to cause a phase response to noise that eventually irritates/tires the user and makes them want to switch the radio off. In the case of the noise response of the DDC SDR Mercury, it was balm to the ears of VK6APH, VK6VZ and VK6GE. The third key advantage of SDRs is that, owing to their signal processing and selectivity being provided digitally by a computer rather than crystal filters, you can provide continuously variable selectivity down to a few tens of Hertz.

So what are the disadvantages of SDRs for radio amateurs? Well, with the exception of Flex-Radio's FLEX-5000 and its predecessor SDR-1000, there are (at the time of writing - early-2009) no HF transceivers in SDR form that are commercially available as an 'off-the shelf' product. However, SoftRock 'sampler' SDR transceivers and receivers are available in kit form which cover parts of one or two HF amateur bands for a few tens of pounds - and these kits can be a heap of fun for any radio amateur with an experimental bent. In addition, the High Performance Software Defined Radio (HPSDR) series of boards allows the technically-savvy to produce a complete state-of-the art SDR transceiver, capable of producing 0.5W for amplification by conventional HF amplifiers. There is much experimentation being carried out which you can read about by joining the SoftRock 40 reflector [13], or the Flex-Radio user reflector [14].

The next disadvantage is that you need a high-end Pentium IV personal computer, preferably capable of a 2GHz-plus clock speed, running *Windows XP*, plus a really good soundcard, often to serve as the analogue-to-digital converter (ADC) for the SDR. For several years, the M-Audio Delta 44 soundcard, which has a sampling rate of up to 96kHz, has been the current *de-facto* soundcard for this purpose. On the other hand, an up-to-date PC is a common household item for most families these days.

The third and final disadvantage is that SDRs force you into a very close relationship with your personal computer, its software and peripheral devices. For those of a Luddite disposition such as VK6VZ, this can be a considerable strain - and even VK6APH has had the odd impulse to throw his PC through the shack window - but the benefits of SDR are too great to ignore.

The radio amateur of today can be working on household tasks on their personal computer, such as e-mailing relatives or doing the family budget on a spreadsheet, whilst leaving a 'window' open on the screen literally to keep an eye on the state of their favourite band using an SDR's bandscope facility.

SDR WITHOUT A RADIO

The good news is that you don't need a SoftRock or an SDR-1000 to experience the performance that an SDR offers - you can try out / emulate an SDR on your personal computer.

Since the audio presented to the soundcard associated with PC-based SDRs is in the range of 0 - 48kHz, the I and Q signals from the 'front end' of an SDR radio can be recorded as standard .WAV files. Such .WAV files can then be simply played back with SDR software such as *Rocky* and *PowerSDR*^{TM}, as if the user were actually connected to the hardware. By playing back I and Q .WAV files in this manner, radio amateurs can experience others use of SDRs. It is possible to download and experience .WAV files of moonbounce contacts, microwave rain scatter contacts, meteor scatter contacts and listen to what 7MHz sounds like in the USA during a major contest [15].

DESCENDANT OF THE DC RECEIVER

Radio amateurs have had a love-hate relationship with the direct conversion or 'DC' receiver for many years - and in essence many SDRs, in particular those with a quadrature switching detector (QSD) architecture (of which more later) - use direct conversion principles. Whilst we love the simplicity and the pure sounds that direct conversion receivers produce, the poor unwanted image rejection of most designs has limited their mass appeal as a communications receiver.

Whilst it is relatively easy to get 40dB of unwanted image suppression with a direct conversion receiver, obtaining the level of image suppression typically achieved by a receiver using superheterodyne principles has historically proven far more difficult.

Some direct conversion receiver designers have suggested that DC receivers should be run without an automatic gain control

The four lines of code that changed the SDR world

I/Q balancing in VE3NEA's *Rocky* software

Balancing of the phase/amplitude of the I and Q signals in the *Rocky* software to maximise image rejection in the Softrock 40 receiver is carried out automatically and is around 90dB. This technique developed by the author of *Rocky*, Alex Shovkoplyas, VE3NEA, has revolutionised the way image rejection is carried out in SDRs.The explanation below is taken from VE3NEA's web pages on Rocky at [1.

"I/Q balancing in Rocky is automatic and does not require any lab equipment – all you have to do is to start the program when the band is open. *Rocky* will use all the strong stations on the band as signal generators!"

"The algorithm works as follows. The power spectrum is scanned for signals that are at least 30dB above the noise. For each signal, synchronous detection of the image is performed using the main signal as a reference oscillator. The synchronous detector has very high sensitivity and can detect the image signal even if it is below the noise. For the signal in the j-th bin of the spectrum, the normalised out impedance of the detector is calculated as follows:

```
Z := ComplexMul(ASpectrum[j], ASpectrum[FftSize-j]);
   Pwr := Sqr(ASpectrum[j].Re)+Sqr(ASpectrum[j].Im)+ Sqr(ASpectrum[FftSize-j].Re)+Sqr(ASpectrum[FftSize-j].Im);
   Z.Re := Z.Re / Pwr;
   Z.Im := Z.Im / Pwr;
```

"The Z value is complex and contains information about the amplitude and phase of the image in respect to the main signal. The program averages Z over time, calculates the required amplitude and phase correction as a function of frequency, and fits a polynomial to the correction coefficients. The I/Q correction filter is constructed from coefficients and applied to the input signal in the frequency domain."

(AGC) circuit, in order to hide the poor image suppression. Whilst this can help, it is necessary to continually 'ride' the volume control to prevent being deafened when tuning from a weak to a strong signal - hardly a pleasant experience.

Whilst the introduction of digital signal processing (DSP) techniques to direct conversion receivers can add a degree of improvement to them by replacing the analogue audio filters used to provide their selectivity with much sharper DSP filtering [16], the problem of the lack of unwanted image rejection has remained - until the development of SDR technology.

To give you an idea of how difficult it is to obtain good image suppression with direct conversion receivers, an accuracy of one degree and 0.1dB in the relative phase and amplitude of the I and Q signals arriving at the input of the demodulation stage is necessary to obtain 40dB of image rejection. In order to obtain 60dB of rejection, the in-phase (I) and quadrature (Q) signals produced by the first stage of a QSD-based SDR - see Fig 8.4 - need to be within 0.1 degree of phase and 0.01dB of amplitude of each other! Even then, 60dB of image rejection is a long way short of what is obtainable from a superheterodyne receiver that uses a crystal filter in its IF amplifier chain. Traditionally, the better DC receivers in this regard have used crystal oscillators that operate at four times the desired reception frequency, driving dual digital dividers to give two local oscillators with a 90° phase difference from each other. This mostly takes care of the requirement for an accurate 90° phase shift between the I and Q signals provided by the local oscillators.

Unfortunately, this still leaves the problem of obtaining/ matching the phase and amplitude of the received signals that appear at the antenna. Even if the local oscillator I and Q balance is perfect, there are still phase shifts present, caused by the antenna, RF amplifier/preamplifier circuits, band pass filtering, etc. Such amplitude and phase variations can be quite considerable over the width of an amateur band and have a seriously detrimental effect on the direct conversion receiver's image rejection/performance.

QSD-based SDRs attempt to overcome this major flaw in a number of ways. If used with the *PowerSDR*^{TM} software, the latter provides up to 90dB of unwanted image rejection by allowing the user, automatically or manually, to trim the amplitude and phase of the incoming I and Q signals in software. As successful as this approach is, it really only works over very narrow bandwidths - typically three to four kilohertz - before it has to be adjusted again. As a result, the image rejection requires readjusting as the receiver tunes over an amateur band - and when it is switched from band-to-band.

During a 'TeamSpeak' forum [17] conducted by enthusiastic users / designers of the SoftRock experimental SDRs which are QSD based, VK6APH suggested there was a way around this problem. Rather than the tradition of using a crystal oscillator at four times the desired reception frequency for DC receivers/SDRs, a crystal oscillator should be used at the reception frequency, followed by a simple resistance-capacitance (RC) phase network to provide an approximate 90° phase shift. He reasoned that whilst this was not ideal and would provide significantly greater variation in I and Q phase accuracy over an amateur band, having an oscillator operating at/close to the reception frequency could be useful to solve this and other problems.

To optimise the 'approximate' I and Q amplitude phase accuracy provided by the RC network, VK6APH suggested that a 'database' or a form of look-up table could be used to store what the 'optimum' I and Q amplitude and phase were for the SDR every few kilohertz. The optimum phase information in the database/lookup table could then be used to correct the approximate information provided by the RC network and provide a very high degree of unwanted image suppression over the width of an amateur band. In order to set-up the look-up table, it would be necessary to sweep a signal generator over the passband of the SDR, using a number of spot frequencies, and store the resulting I and Q amplitude and phase values.

At around the same time, VE3NEA, author of a number of well-known and respected amateur radio software programs, came up with a much better solution to the problem of I and Q amplitude and phase correction in QSD-based SDRs. Instead of using a signal generator to generate a set of signals, he used actual signals within the amateur band that the SDR was tuned to automatically, and continuously update the look-up table to meet the I and Q balance requirements.

VE3NEA pioneered this 'intelligent' technique in his free *Rocky* software [1]. This technique reaches the 'holy grail' of DC receivers of achieving around 90dB image suppression - comparable to a very good superhet - by means of a few lines of software (see box). VK6APH reverently refers to these lines as "the four lines of code that changed the SDR world."

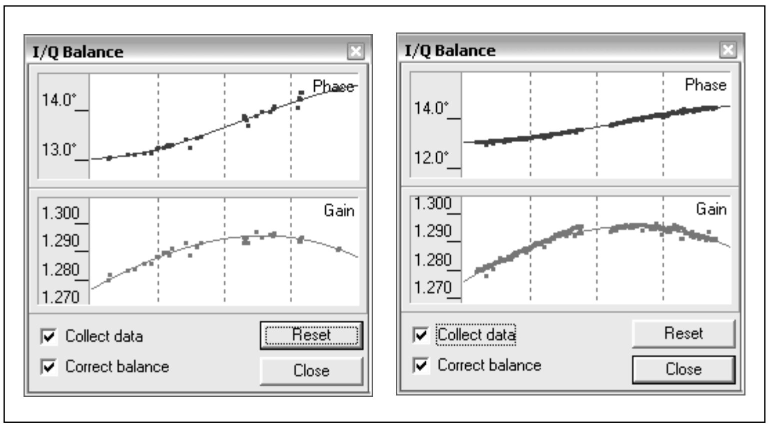


Fig 8.6: The I and Q balance of a SoftRock receiver with *Rocky* software at switch-on (left) and after a day

Fig 8.6 shows the I and Q balance of the SoftRock 40 SDR fifteen minutes after first being switched on, and the much improved I and Q balance of the SoftRock 40 the following day. As far as we know, this is the first example of a receiver that actually analyses and dynamically optimises its performance during operation. The longer you leave your SoftRock receiver and *Rocky* on, the better the image rejection gets.

Other radio amateurs have been quick to make use of VE3NEA's revolutionary technique. Duncan Munroe, M0KGK, has used it in his free *KGKSDR* software for SDR radios.

VK6APH's gentle goading of the mathematical Bob McGuire, N4HY, appears to have born fruit in that Bob has recently announced a fast and automatic way of compensating for I and Q phase errors in *PowerSDR*^{TM}. Bob reports that this is achieved in a mere 20 lines of code. We can expect this breakthrough to be rapidly adopted by these writing software for QSD-based receivers.

QSD VERSUS DDC

In this section we are going to compare the characteristics of the two most-popular SDR receiver architectures.

The first - see Fig 8.4(a) - is the architecture most frequently used by amateur radio designers and is based on a quadrature switching detector (QSD) followed by an analogue-to-digital converter (ADC) based either on a sound card or chip. The highly popular SoftRock [2] series of receivers and transceivers and the Flex-Radio SDR-1000 and Flex-5000 uses this design.

The second architecture - see Fig. 8.5 - is that of the emerging digital down conversion (DDC) architecture, based upon on a high-speed ADC connected directly to the receiver antenna - as you will have read earlier in this chapter, this is the architecture favoured by VK6APH and VK6VZ. Examples of receivers using this technique are the Perseus [5], SDR-IQ [6], Hans Zahnd HB9CBU's ADAT ADT-200A [7] and the HPSDR Mercury [8].

Both of these architectures have their advantages and disadvantages - some of these for each design are listed against a number of key SDR receiver design parameters in Table 8.1. Let's look at how they compare.

I & Q Balance

In order to eliminate the unwanted sideband, the amplitude of I and Q signals - and their 90-degree phase relationship - must be held to very high tolerances. For example, to achieve 60dB of sideband (or image) suppression, the I and Q signals must match within 0.01dB (of amplitude) and 0.1 degrees (of phase).

For an analogue-based design like those using QSD, this is a tall order - particularly over the entire HF range. In practice, there is a need to resort to some form of compensation to attain this level of sideband/image suppression, usually done in software as described earlier.

Other techniques based on single frequency correction, either manual or automatic, are able to achieve very high levels of suppression. However, such correction is often only effective over a narrow frequency range and can become unacceptable at band edges.

This is one area where a lot of innovative research is going on and we can expect to see some exciting new techniques to solve this problem in the near future.

Sideband (image suppression and carrier suppression) is simply not an issue with DDC-based receivers since the I and Q signals are generated mathematically and for practical purposes this suppression can be considered to be perfect.

Local Oscillator Radiation

This item relates to unwanted radiation from the antenna socket of the SDR receiver's local oscillator. With a QSD-based analogue SDR receiver, the local oscillator frequency is close to the receiver's actual frequency and thus is likely to find its way to the antenna and be radiated. Measurements on such receivers can yield signals at their antenna socket of -50dBm - and these will be received by local stations. However, the addition of a preamplifier can substantially reduce the level of such radiation.

Since the local oscillator in a DDC-based SDR receiver only exists in the form of multiplication by digital values inside an integrated circuit, it is not normally detectable at the antenna socket. This does not necessarily mean that no internal signals find their way to the antenna socket, since there will be a host of clocks used by the DDC. However, with careful lay-out of the DDC-based receiver components, the issue can be minimised.

Spurious Signals

Both types of SDR receiver architecture can present low-level spurious signals due to the local oscillator used, as the receiver is tuned across a range of frequencies. Since the SoftRock receivers are based upon a spectrally-clean crystal oscillator, these can be expected to produce either a very few spurs - or none - over their relatively narrow tuning range.

SDR receivers based on currently-used direct digital synthesis (DDS) chips (eg the popular Analog Devices AD9851) will exhibit numerous low level spurs. However, the latest DDS chips generate significantly fewer spurious signals than their predecessors and, on the lower HF bands at least, those spurs that do exist are frequently masked by band noise. The use of a preamplifier on the higher HF bands will assist with spurs, but this also reduces the large-signal handling capability of the receiver.

Since DDC-based receivers do not require a digital-to-analogue converter (DAC) as part of a DDS to generate a local oscillator signal, a significant source of spurs is removed. In addition, it is possible to use a higher number of bits to synthesise the local oscillator signal - typically 18 as opposed to 14 - for a DDS chip, which again reduces the number of spurs.

Experience to-date with the range of DDC receivers currently available indicates that local oscillator spurs are not a problem in practice. In the case of the Perseus and HPSDR Mercury receivers, the spurs are below -105dBc.

Both DDS and DDC-based SDR receivers are ultimately reliant on the stability and spectral purity (ie phase noise) of their master clocks - which is why there is so much interest among SDR designers in having the best possible master clock.

Frequency Coverage

In general, QSD-based receivers are usually effective on the amateur bands from 1.8 to 52MHz. Whilst in theory the QSD should perform effectively into the microwave bands, there appears to be a technical issue with turn-on and turn-off times of the QSD's solid-state switches that limits their performance above 70MHz. The issue results in a noise figure - but not necessarily conversion loss - that increases with frequency. This is generally overcome in practice by using a low-noise preamplifier before the QSD. However, although this reduces the receiver's noise figure, it detracts from the receiver's large signal handling performance. We will have more to say on QSD performance later in this chapter, as there are some interesting developments in this area.

The frequency coverage of a DDC-based SDR is basically restricted by the sampling rate of the ADC, ie its clock speed. Sampling theory provides us with a range of 0 to clock/2 MHz. This means that for a modern 14 or 16-bit ADC clocked at 135MHz, continuous coverage up to 67.5MHz can be obtained. In practice, life becomes difficult as operation approaches the actual clock/2 frequency, but should prove quite practical up to, say, 67MHz. With suitable filtering, signals can also be received from clock/2 to the clock speed - this is the so-called 'alias response'. Such a receiver will normally tune in reverse (and invert an SSB signal) but fortunately there are some simple software tricks that can be used so that the user is unaware of this issue.

Given a high performance ADC, repeating alias responses can be used to receive signals as high as upper UHF. For example, the ADC used in the HPSDR's Mercury - a Linear Technologies' LT2208 - will accept inputs up to 750MHz. However, in general, the higher the frequency, the lower the sensitivity of the ADC becomes. With a 125MHz clock, Mercury will receive strong local signals on 144MHz 'as is' and, with suitable filtering and pre-amplification, provides a very acceptable performance.

Harmonic and Sub-harmonic Responses

The QSD detector will respond to signals at odd harmonics of its clock frequency - which can be both a blessing and a curse! This effect is used to good effect in the SoftRock range of receivers and transceivers, so that lower frequency crystal oscillators can be used on the higher HF bands.

In order to eliminate high harmonic responses, a high-performance low pass filter (LPF) is required prior to a QSD. The latest FLEX-5000 uses an 11th order filter to provide over 70dB of suppression on all amateur bands.

Whilst in theory the QSD should not respond to sub-harmonics of the local oscillator frequency (in fact US mathematician Bob, N4HY, strongly supports this theory), a number of experimenters, including VK6APH, have noticed this can happen in practice. Further investigation appears to be in order.

Assuming the circuitry prior to the ADC is perfectly linear, DDC-based receivers will not have harmonic responses. However, maintaining linearity over a 125dB blocking dynamic range (BDR), in particular for sub-harmonic signals, is a significant design challenge. In particular, high pass filters (HPFs) in the signal path need to use carefully selected inductors. This can lead to the use of large inductors that would normally look more at home in a transmitter LPF! There is more on this subject in the later section devoted to the HPSDR Alex HPF/LPFs.

RF Bandwidth

This is a feature that really shines in the QSD architecture. The circuitry forms an LPF consisting of the antenna impedance, switch losses and any other series resistance prior to the sampling capacitors. This LPF will typically have a 3dB bandwidth of 3kHz (assuming a bandscope is not being used, in which case 48/96/192kHz would typically be used). Since this bandwidth appears on both sides on the local oscillator signal, it will appear as an RF bandwidth of 6kHz. For a QSD-based receiver operating at, say, 7MHz, this represents a Q of approximately 2,300. If this is extended to 28MHz, the Q increases to 9,300 - a value that will be extremely difficult to match with a conventional LC filter. Since this Q is before any ADC, it offers a significant level of attenuation to out-of-band signals.

The current crop of DDC-based receivers tend to be wideband, covering from a few tens of kilohertz right up to 50MHz or higher. Bulk filtering in the form of a high order LPF, to remove VHF aliasing signals, is the order of the day. A number of designs are using half-octave band pass filters. The effect of large out-of-band signals is not necessarily so much of a problem with DDC-based receivers and at least one manufacturer [18] is initially offering a receiver with just a 50MHz LPF at the input.

Intercept Point

Again this is an area where the QSD-based receiver has shone for some time. Assuming the QSD is feeding a correctly-designed post amplifier and a high-quality sound card, then IP3 values of over 30dBm are common. The *RadCom* review of a SoftRock v6 receiver by Peter Hart [19] showed an excellent IP3 of +19dBm (or about 93dB dynamic range in SSB bandwidths). The use of IP3 when applied to DDC-based receivers is a controversial subject (as the authors have found out in the past when innocently publishing the IP3 figures for the DDC HPSDR Mercury receiver [20] in a *RadCom* column). However, we are also first to admit that IP3 measurements on such receivers are not meaningful, since the values obtained vary with the input signal levels.

Given that there are a number of different designs of DDC receivers now in radio amateur hands, we can expect to hear shortly if intermodulation issues are going to prove a problem. In the case of the HPSDR Mercury receiver, it has a superb IP3 performance: +33dBm (with the preamplifier on), rising to over 50dBm (with the preamplifier off). Note the IP3 of Mercury is independent of tone spacing.

Noise Figure

The theoretical noise figure of a QSD is approximately 1dB. If it is assumed that the combined noise figure of the post amplifier that follows it and the sound card is also 1dB (which should be achievable using modern integrated circuits) then an overall noise figure of 2dB should be obtainable. This is well above what is practically required on any of the HF bands and should serve well into the VHF region. However, there appear to be some issues in actually achieving this level of performance.

DDC-based receivers are currently yielding noise figures in the region of 30dB. Although this sounds high, such a value would be usable on the bands from 1.8MHz through to 10.1MHz without a preamplifier. On the higher bands a preamplifier will be required in order for those operators blessed with a low noise level location to reap the benefits.

As ADC technology continues to improve, it is expected that future generations of these devices will be usable on the HF bands without the need for a preamplifier.

Cost

If it is assumed that both the QSD and DDC-based receivers use a PC, together with one of the free SDR software programs (such as *Rocky*, *PowerSDR*^{TM} or *KGKSDR*), for signal processing then it is reasonable to remove this cost from the comparison.

Again, the QSD is a winner. The cost of a complete RF front-end for a QSD-based single band receiver (eg a SoftRock v6 lite) is in the order of U$10 to US$13, increasing to US$30 to US$51 for a QSD-based single or dual-band transceiver. However, in order to achieve the full performance of the QSD we need to use a high-quality sound card in the PC such as the M-Audio Delta 44, which (at the time of writing) will typically cost around US$150.

The high cost components in the DDC-based receiver are the ADC and a high-performance VHF clock. Currently the most popular ADC for amateur radio applications is the Linear Technology LT2208, which in low volume costs about U$100. A suitable 'clock' will cost about US$50. Granted, these components are not inexpensive, but given that a high performance sound card is not required then the cost appears reasonable. The LT2208 ADC is proving to be very popular, so it is possible that high sales volumes will cause a price reduction in future.

Complexity

Once again, the QSD has the edge. A handful of common components will produce a receiver front-end that will compete with the best of current analogue designs. Even some of the more advanced QSD designs do little to increase the complexity.

Whilst a DDC receiver, based on an ADC plus Field Programmable Gate Array (FPGA), has a low component count, the internal code required to suitably process the ADC data is quite complex. Indeed, the Verilog code used in the HPSDR Mercury DDC took VK6APH the best part of six months part-time programming to develop.

However, given that much of this code is now in the public domain (under the GNU General Public Licence) then those who wish to develop their own variant have a significant head-start.

OPTIMISING AN SDR BANDSCOPE

By using a standard personal computer to undertake the processing for an SDR, we obtain the significant benefit of a spectrum scope or bandscope.

Bandscopes are certainly not new - they have been used for several decades by both radio professionals and amateurs for monitoring weak signals. Heathkit made a bandscope you could add onto a receiver in the 1970s. However, the new DSP-based bandscopes in SDR radios offer many advantages over their predecessors.

The limitation of the traditional bandscope is that the bandwidth of the filters used is relatively wide. Those that did use narrow communications-style filters scanned very slowly over the frequency bands they covered (to prevent the narrow filter from ringing) and required a long-persistence cathode ray tube.

By using DSP techniques, in particular a Fast Fourier Transform (FFT) to produce a bandscope for an SDR, an FFT 'bin' in the order of 11Hz can be produced. What this means practically is that there is no need to sweep such a narrow filter slowly across the SDR passband, since the FFT applies large numbers of 11Hz-wide filters in parallel to blocks of digital samples.

If we compare the 11Hz filtering possible with an SDR bandscope to that of the 1KHz filters typically used in high-end Japanese radios containing bandscopes, the former offers a much higher sensitivity. In fact, such 11Hz filters enable very weak signals to be seen on an SDR radio which cannot be actually be heard - a very strange phenomenon to experience.

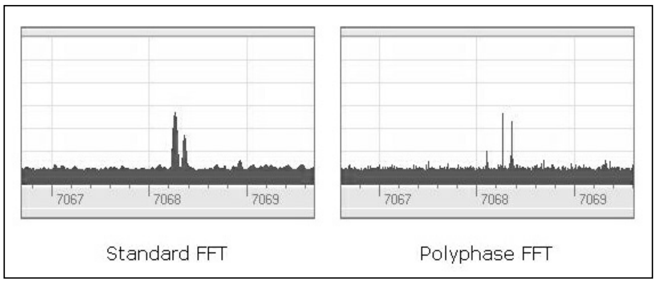


Fig 8.7: The use of polyphase FFT produces better resolution of the bandscope

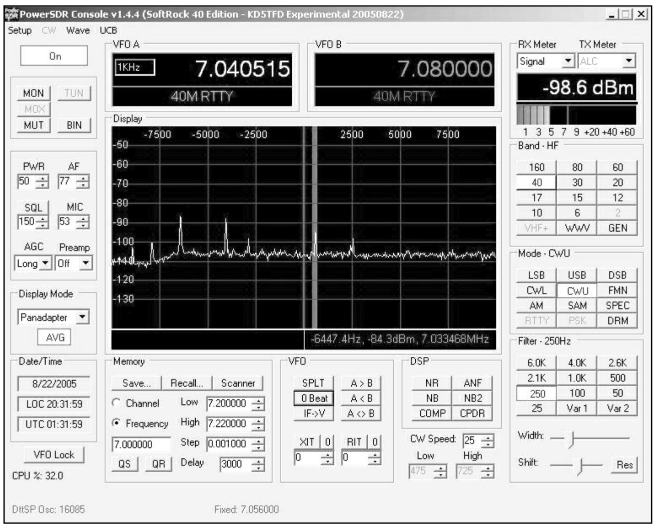


Fig 8.8: Flex-Radio's *PowerSDR™* software, showing the use of polyphase FFT

With the bandscope available in his *Rocky* software, VE3NEA has pioneered the use of a polyphase FFT, which greatly narrows the spectrum that a signal occupies on a bandscope screen, allowing a very clear and sharply defined image to be seen of weak signals. VK6VZ has found that weak 1.8MHz CW DX signals that are only S2 to S3 in strength can be clearly seen on the Rocky bandscope during summer, despite high levels of atmospheric noise.

Fig 8.7 is a screen capture from VE3NEA's website showing the effect of a SDR bandscope that uses a polyphase FFT and one that does not. The use of a polyphase FFT now appears to be almost mandatory for SDR bandscopes and they have been added to both Flex-Radio's *PowerSDR*TM (see **Fig 8.8**) and M0KGK's *KGKSDR* software.

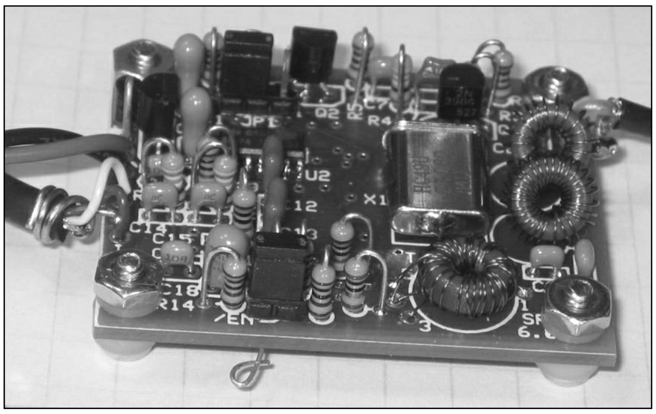
Although a polyphase FFT narrows a signal nicely on an SDR bandscope, VK6APH points out that there is no 'free lunch' since the transient response of the bandscope is substantially increased. For this reason, most SDR software enables the polyphase FFT facility to be turned off. That being said, VK6VZ keeps the polyphase FFT permanently enabled on 1.8MHz because of the improved bandscope performance, despite this band being heavily affected by transient signals/noise.

SDR HARDWARE

Up until 2006, the best way for amateurs to move into SDR was to buy Flex Radio's SDR-1000 transceiver. Whilst reasonably priced, this radio was not inexpensive and what was needed was a cheap 'sampler' SDR to show radio amateurs all the advantages of using an SDR - and see if such technology was for them.

A group of radio amateurs decided to create precisely such a sampler. The SoftRock QSD-based hardware, developed by Tony Parks, KB9YIG, and Bill Tracey, KD5TFD, was the result. These kits, publicised by AmQRP, provide a simple and low-cost way for radio amateurs to evaluate SDR techniques for themselves.

The first versions of the SoftRock hardware were single band 7MHz receivers, modifiable to other frequencies. Subsequently, dual-band receiver and transceiver kits have been produced - see **Fig 8.9** (the transceiver is described in detail later in this



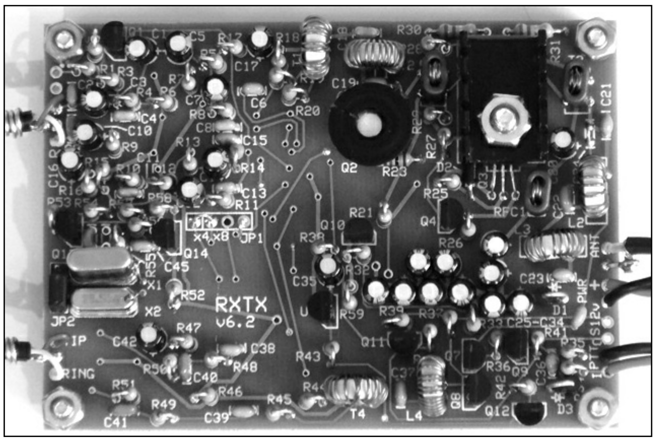


Fig 8.9: A SoftRock version 6 receiver (above) and a version 6 transceiver (below)

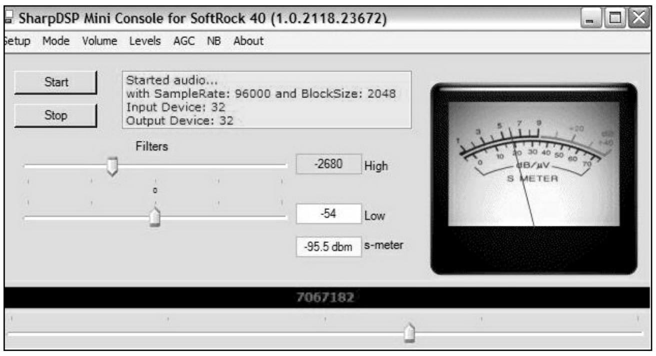
chapter). Tony Parks KB9YIG, produces and markets the SoftRock kits. Note that there is an excellent Internet forum called 'SoftRock 40' hosted by Yahoo which people who are interested in building or developing the SoftRock can join [13].

This range of radios allows you to tune plus/minus half the sampling rate of your personal computer's soundcard from the crystal frequency selected by the user for the SDR hardware. In the case of the M-Audio Delta 44, this is ±48kHz. For example, if the crystal frequency is 7.040MHz, then using a 96kHz sound card sampling rate, the user can tune (using the software packages mentioned earlier) 48kHz either side of this frequency.

Whilst a standard motherboard-residing soundcard is sufficient to evaluate SDR hardware and software, it may have some major limitations in reproducing signals, due to 1/f flicker noise, hum and picking up other sources of interference generated inside the computer casing. The Delta 44 in contrast sits in its own slot well away from the personal computer motherboard and all connections to and from it are in a remotely-sited screened box.

Most motherboard and low-end soundcards are limited to 16-bit, which restricts the dynamic range of the SDR. Using a 24-bit M-Audio Delta 44 connected to a SoftRock Version 4 converted for 1.8MHz, VK6VZ has experienced a dynamic range in excess of 90db - at bandwidths of a few tens of Hertz to 2.4kHz.

The most recent Softrock receivers (and transceivers) are now using the Si570 microwave phase locked loop ('PLL') oscillator chip that can be divided down to give variable frequency oscillator-like performance across all HF bands, removes the limited tuning range issues of earlier designs. This gives a significant advantage over the original method because it also alleviates the need to accurately balance I (in-phase) and Q (quadrature) signal components over a wide bandwidth. There is a section on using the Si570 device with Softrocks at the end of this chapter.


SharpDSP Mini Console for SoftRock 40 (1.0.2118.23672)
Setup Mode Volume Levels AGC NB About
Start
Stop
Filters
S METER
dB/µV

```
Started audio...
with SampleRate: 96000 and BlockSize: 2048
Input Device: 32
Output Device: 32
```

-2680 High

-54 Low

-95.5 dbm s-meter

7067182

Fig 8.10: N8VB's SDR console software

SUPER SOFTWARE DEVELOPMENT - INCLUDING LINUX CONSOLES

The low cost and mass appeal of the QSD-based SoftRock hardware served to spur a number of radio amateurs (such as VE3NEA, M0KGK and I2PHD) to develop free software for it [21].

Flex-Radio's *PowerSDR*TM software, developed for use with the SDR-1000, has been modified to run with the SoftRock range of radios. One of the outstanding features of *PowerSDR*TM is the AGC system. This uses two AGC loops in parallel, one to handle fast signals such as noise pulses or the leading edge of CW signals, and the other to follow the slowing moving envelope of the signal.

In VK6APH's opinion (even allowing for the fact that he designed the AGC system!) when the mathematical algorithm was implemented by Dr Robert McGwier, N4HY, in the *PowerSDR*TM software it resulted in an AGC performance - and resulting clarity of received audio - that was unsurpassed by any other receiver he had heard so far.

*PowerSDR*TM is open source, which has enabled a number of programmers to develop variants of it, for example Phil Covington, N8VB, has produced an excellent piece of SDR console software (see **Fig 8.10**) based on *PowerSDR*TM [22]. N8VB's software was the first to provide dual, independent receivers within the sampling rate bandwidth. Given that each receiver is simply a copy of a block of existing software, it is possible for N8VB to implement multiple (many!) independent receivers in his software.

Others, like Bob Cowdery, G3UKB, have taken totally different approaches. In his SmallTalk project [23], Bob's idea is to create 'building blocks' that can be joined together to allow different configurations of an SDR to be easily constructed and tested.

In addition to *PowerSDR*TM and *Rocky*, there are two other WindowsTM compatible popular pieces of 'free' software available for HF - the first of which is Duncan Munroe, M0KGK's *KGKSDR* software [3], which was touched on briefly in the introduction. Like VE3NEA's *Rocky* software, *KGKSDR* was designed to work primarily with the SoftRock range of receivers and transceivers. However, whilst *Rocky* is optimized for weak signal CW DXing and PSK31, *KGKSDR* is designed for a wider range of modes/uses and has more user-selectable parameters.

In many ways, the *KGKSDR* bandscope display - see **Fig 8.11** - has the best of both worlds of the *Rocky* and *PowerSDR*TM band-scopes. Like *Rocky, KGKSDR* has intelligent I and Q balancing, enabling automatic image suppression, and polyphase Fast Fourier Transformation, giving an extremely high resolution bandscope. However, like *PowerSDR*TM, you have the choice of several different AGC speeds, to suit the particular mode you are using - the software allows CW, SSB, AM and PSK use; plus five switchable CW receive bandwidths, varying from 100 to 500Hz.

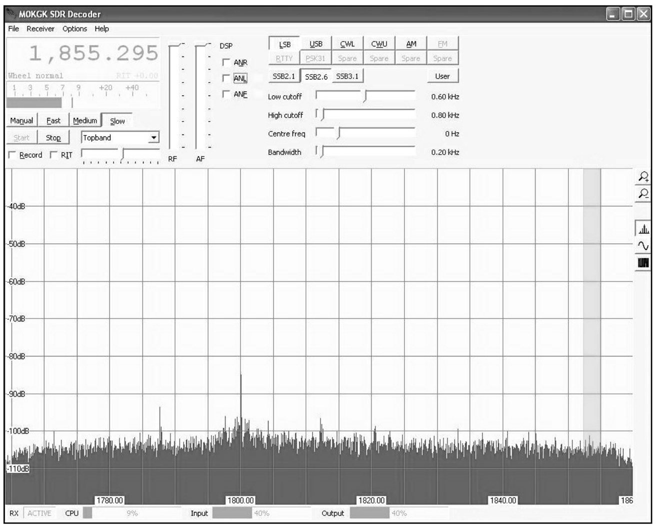


Fig 8.11: Screenshot of the *KGKSDR* bandscope display

Winrad is written by Alberto di Bene, I2PHD, with assistance from Jeffrey Pawlan, WA6KBL, a highly respected EME operator. It is specifically tailored for weak signal SSB and CW reception and can be downloaded free from Alberto's website [24]. This web site has two excellent samples that show the effectiveness of *Winrad*. One sample is an EME CW signal made completely unreadable by static crashes. In the second sample, the same signal is rendered decipherable by being filtered through *Winrad*. Alberto has also developed another experimental SDR application for free download called *SDRadio*, which can demodulate modes including SSB, AM and FM.

For the experienced radio amateur who enjoys 'cutting code', there are numerous opportunities to experiment with new SDR ideas in software form. For the programmer who wants to play around with SDRs, there are a number of options depending on your skill level. For the relative beginner, planned changes to the open source *PowerSDR*^{TM} code, that will result in digital signal processing (DSP) part of the software becoming separated from the graphical user interface (GUI) part, mean that they have an opportunity to develop their own GUI, in the programming language of their choice.

Those who have the know-how to work in the bowels of *DttSP* - the open source software written by Dr Frank Brickle, AB2KT, and Dr Robert McGwier, N4HY, that commonly implements the core DSP functions of a SDR - are already writing their own GUIs or 'consoles', often running under the Linux operating system or one of its variants.

DttSP implements "the basic modulation, demodulation, signal conditioning and synchronization processes required to operate a high-performance transceiver using DSP as the detection and synthesis stages" [25]. *DttSP* is actually the DSP core/engine that powers *PowerSDR*^{TM} and its development is carried out primarily on Linux.

To see how Linux software development for SDRs is done, let us look at the 'consoles' developed by John Melton, G0ORX/N6LYT, Edison Pereira, PU1JTE/N1VTN, and Rob Frohne, KL7NA. A search of the internet will reveal other examples of Linux consoles, but these are some of the best known and most recent examples.

That being said, by far the most famous of all Linux software for SDR and the grand-daddy of them all is Leif Asbrink, SM5BSZ's amazing *Linrad*, but this is effectively a stand-alone SDR program of its own. *Linrad* can be run under Linux as well as under Microsoft Windows^{TM} on an IBM PC-compatible computer. It is available as source code from SM5BSZ's website [26] with a 'makefile' that allows the generation of an executable with a single command, both under Linux and Windows.

Linrad can operate with any soundcard for which the operating system on the PC has suitable driver routines. A conventional radio receiver or a direct conversion radio is then used to bring some part of the RF spectrum down to audio frequencies.

The DSP core in *Linrad* software is independent of the hardware and will processes any bandwidth that the latter can handle. The origin of *Linrad* lies in software that was developed by SM5BSZ for the reception of 144MHz CW during EME working. Its noise blanker is to die for.

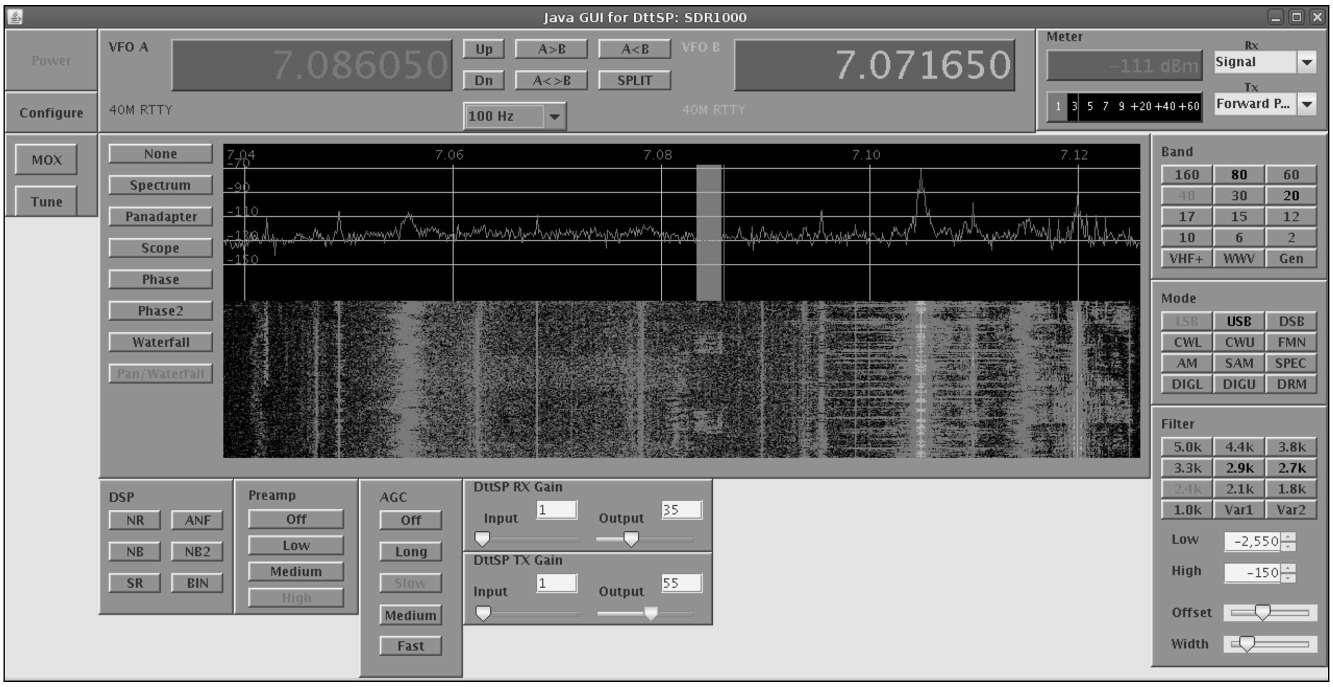


Fig 8.12: Linux console software for Flex Radio's SDR-1000 by John Melton, G0NRX/N6LYT

John Melton, G0NRX/N6LYT's Linux console - see **Fig 8.12** - was actually written in the Java language to control the Flex Radio SDR-1000 under Linux. Although most people with SDR-1000s use the *PowerSDR*TM software running under Windows, there are several experimentally-minded software-orientated SDR-1000 users who wish to use the open source Linux software. John has decided to support the HPSDR project and news of his progress, via his internet 'blog' [27], makes interesting reading.

Linux, as its name suggests, is a Unix-like computer operating system and one of the classic examples of open-source software development - typically, all its underlying source code can be freely modified, used, and redistributed by anyone. The Linux operating system was developed by Linus Torvalds in 1991 and its utilities and libraries usually come from the GNU operating system, which goes back a further eight years

G0NRX's console will also run under Linux-like operating systems such as Ubuntu [28] and Apple's Mac OS X [29]. John's console is intended to control the SDR-1000, Softrock and HPSDR on both receive and transmit, but is still under development. As of April 2009, the console supports 192K, 96K and 48K sampling, includes a waterfall display and, at present, is essentially for reception use.

In his own words, Edson, PU1JTE /N1VTN's SDR-Shell (see **Fig 8.13**) is "a simple SDR GUI for controlling the *DttSP* sdr-core" and can be downloaded from his website [30]. It was implemented to be used with simple SDR receivers like the SoftRock series [2] and SDRZero [31]. SDR-Shell was developed using the *Qt Toolkit* under the Ubuntu flavour of Linux. The transmit functions are currently being implemented to enable transmit on simple SDR transceivers like the SoftRock RXTX 6.2. A block diagram of Edson's popular and flexible SDR-Shell software is given in **Fig 8.14**.

Rob Frohne, KL7NA, has used PU1JTE/N1VTN's SDR-Shell as the jumping-off point for his special version of the latter - one of the main ideas behind open-source software is that a software writer can take what someone else has written and adapt it for a different purpose. In the case of KL7NA, Rob's aim was to produce SDR software that he could run under the Ubuntu flavour of Linux - see **Fig 8.15** - in order to give his TS-850SAT receive section an add-on sensitive bandscope and a DSP receiver back-end (that he can listen to, rather than listening to audio from the TS-850SAT).

Rob tapped the 455kHz third intermediate frequency (IF) chain in the receive section of the TS-850SAT and fed this output into a SoftRock receiver modified for 455kHz, so as to provide DSP and a bandscope display using his specially modified version of *SDR-Shell* and digital-to-analogue conversion using his PC's soundcard. According to KL7NA's web pages [32], which give a full description of his *SDR-Shell* project, Rob used the Synaptic graphical management package to get the low-latency kernel, then used Roger Rehr, W3SZ's excellent *DttSP* and Linux help pages [33] and the *Debian* package [34] to get *DttSP* and *SDR-Shell* going on his computer, which was running the Ubuntu Feisty (7.04) operating system.

KL7NA says the *SDR-Shell* script had to be modified a little and that he learnt a lot about how the *DttSP* system worked with first-in first-out (FIFO) memories and the JACK audio connection kit [35] that he used by reading the shell script and "bothering the experts on the DttSP-linux mailing list" [36].

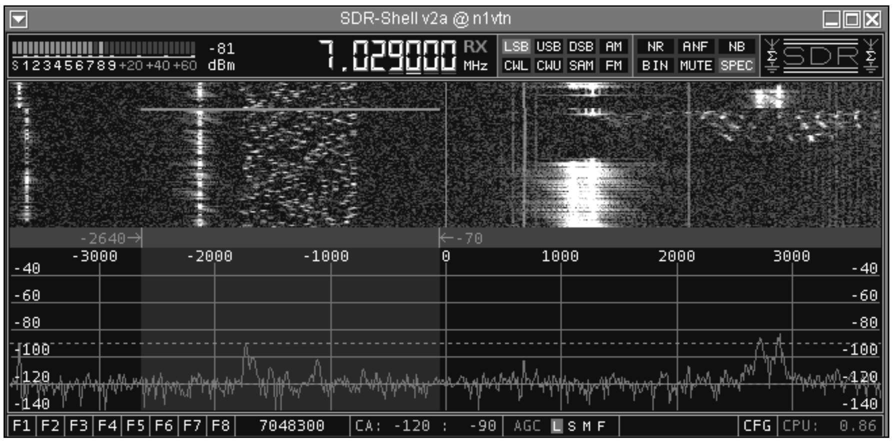


Fig 8.13: Edison PU1JTE/N1VTN's *SDR-Shell*

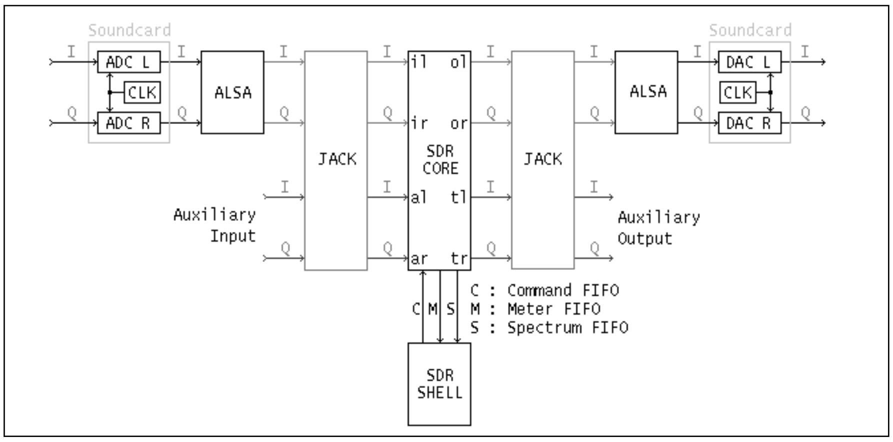


Fig 8.14: Block diagram of PU1JTE/N1VTN's *SDR-Shell* Linux console (from his website)

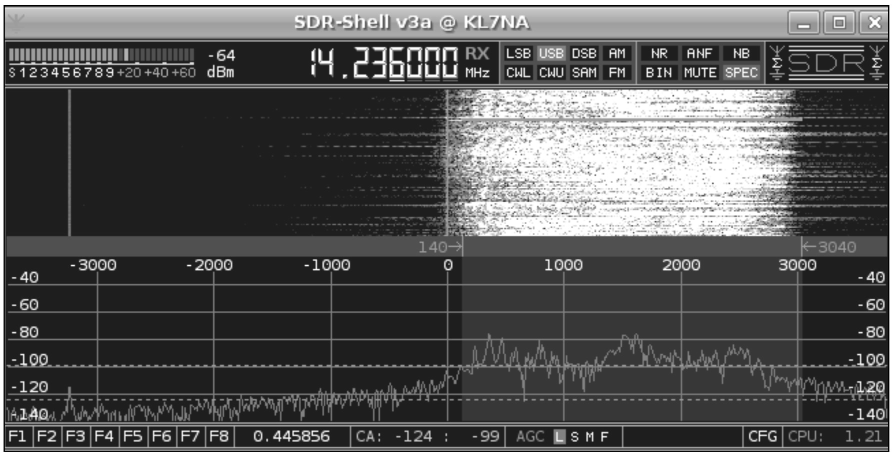


Fig 8.15: KL7NA's version of *SDR-Shell* for receiver IF applications

KL7NA's SDR-Shell/SoftRock set-up with his TS-850SAT enables him to look at around 12kHz of spectrum at a time. It also provides Rob with synchronous AM detection, automatic notch filtering and binaural audio. Rob's web pages provide details of how he modified his TS-850SAT and his version of PU1JTE/N1VTN's SDR-Shell can be downloaded from there.

INTRODUCING THE HPSDR PROJECT

The High Performance Software Defined Radio (HPSDR) is an ambitious project being carried out cooperatively by a group of radio amateurs. Its ultimate aim is to make an SDR that covers all the bands from 160m to 6m, with a performance that is superior to existing analogue and SDR radios.

The HPSDR is a hardware and software project based on the open-source GNU concept. First of all, a bit about the oddly-named GNU (pronounced 'noo' or 'new') project - this originated in 1984 to develop a complete and free UNIX-like operating system. Variants of this GNU operating system, which use the Linux kernel, are now widely used - such in the *GNURadio* described in the previous section - and one of these will run the HPSDR in addition to Microsoft Windows™.

The HPSDR is being designed and developed by a multinational group of SDR enthusiasts - including VK6APH. The membership of the web-based group is well over 100 and includes other well-known experimentally-minded radio amateurs such as Phil Covington, N8VB, Lyle Johnson, KK7P, Graham Haddock, KE9H, and Bill Tracey, KD5TFD. The project has a dedicated web site [8] and wiki which explains the basic ideas behind the HPSDR and how to subscribe to the project's e-mail reflector.

The rationale behind the HPSDR project is to break the overall design of a high performance SDR up into a number of self-contained modules. Each module is designed by an individual or small group and connects to the other modules using a predefined and common bus-type 'backbone' - in a similar manner to plugging boards into a PC motherboard. As the HPSDR project is 'open source', the circuits of each module and PCB design files, and any software they use, are made freely available on the web.

One of the first modules produced was the Janus board, which replaces a PC soundcard as an analogue to digital converter and is being used to improve the performance of the Flex-Radio SDR-1000 transceiver and SoftRock receivers and transceivers. However, its major achievement is the production of three modules - the Mercury Digital Down Conversion (DDC) receiver, the Penelope Digital Up Conversion (DUC) transmitter and the Ozy SDR to PC communications board - which together form the basis of a high-performance 0.5W digital SDR transceiver.

This modular open source approach allows radio amateurs to incorporate just the modules that interest them into their own personal HPSDR. They can also design their own variants of the modules. The approach encourages the development of new ideas, circuits and techniques. One key philosophy behind HPSDR is for its builders/users to be able to replace the digital modules contained within it as better digital devices, such as analogue-to digital converters (ADCs), become available, but to retain those analogue modules, such as RF bandpass filters, where the technology is unlikely to improve in any significant way.

This is because in a couple of year's time, it is hoped there will be an ADC available that will offer significantly better performance at an affordable price than the Linear Technology LTC2208 130Msps 16-bit ADC that the current version of Mercury uses. On the other hand, inductance-capacitance (LC) bandpass filtering technology is unlikely to change, so this is better kept on a separate board (such as HPSDR's Alexaires module) to the receiver ADC.

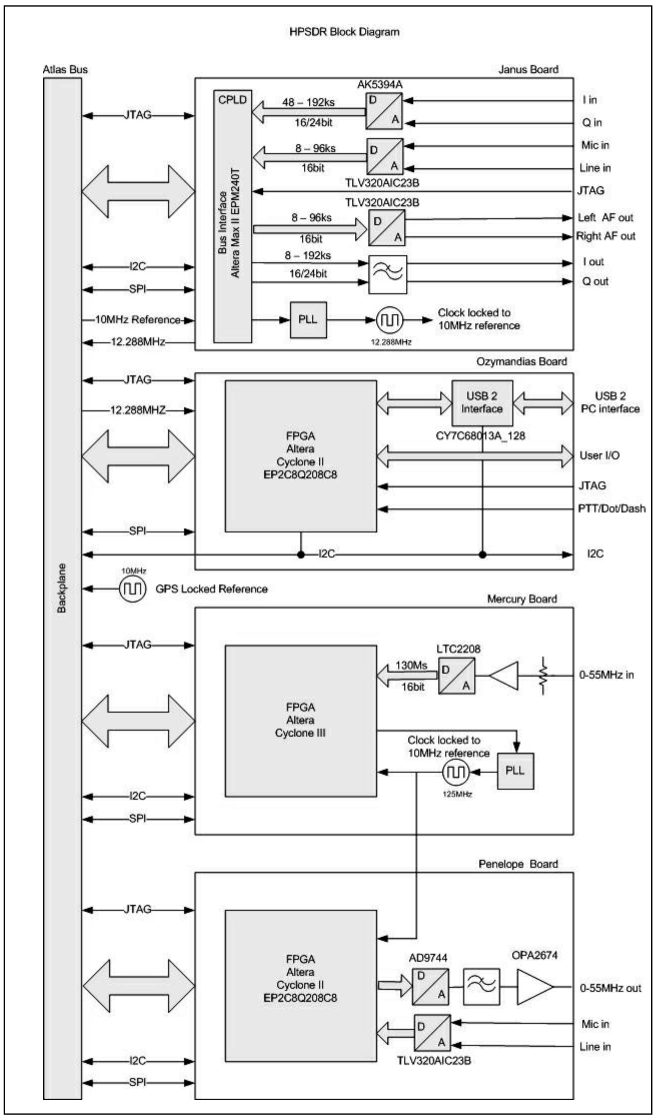


Fig 8.16: Block diagrams of the main current HPSDR modules

The HPSDR modules vary considerably in complexity - from simple bandpass filters and input/output interfaces to full-blown digital signal processing (DSP) functions. Such a variety of modules and complexity enables experimenters - both experts and novices - to make a genuine contribution to the project. Some of the HPSDR modules are being designed so that they can be either used in conjunction with others or in a stand-alone manner. Most of the main modules are shown in **Fig 8.16** and have the following functions and features.

Atlas - the Backplane

This HPSDR module consists of a passive backplane / backbone that all the other HPSDR modules will plug into - see the picture of an Atlas board in **Fig 8.17**.

The circuit board carries six DIN41612 connectors. In addition, an ATX 20-pin power connector is fitted to the board so that 12V, 5V and 3.3V supplies from a standard PC power supply can be used to power the HPSDR. Since such power supplies are readily available, both new and on the surplus market, this neatly solves the HPSDR's power requirements. The DIN connector spacing and board size have been chosen such that the backplane can be fitted into a standard PC enclosure.

The Atlas board was prototyped back in March 2006 and, if available, can be ordered from the not-for-profit Tucson Amateur

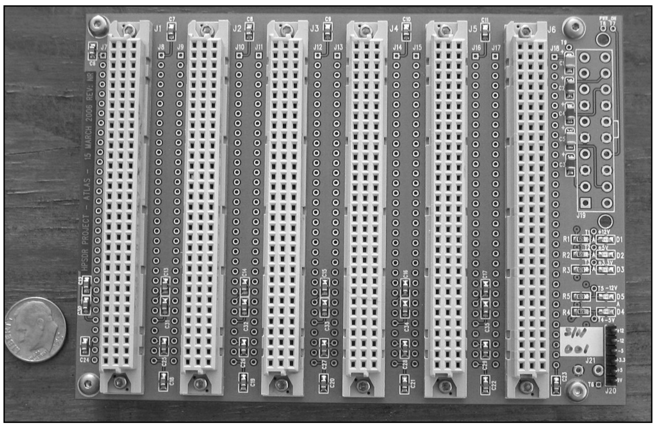


Fig 8.17: Prototype HPSDR backplane/backbone [Photo: Phil Covington, N8VB]

Packet Radio organization [10]. The various files for the design/construction of the Atlas board can be found on the Internet [37].

Note that each module for the HPSDR has a project leader - just like in the development of a commercial electronics product. For example, in the case of the Atlas module, this was Phil Covington, N8VB. Eric Ellison, AA4SW, is overall project manager for the HPSDR and his job is to work with each of the project leaders to foster and support the development of a full-blown HPSDR.

Janus - the DAC and ADC board

Named after the mythical god who can look in opposite directions simultaneously, the module is a very high performance, dual, full-duplex, analogue-to-digital converter (ADC) and digital-to-analogue (DAC) converter board.

Whilst the M-Audio Delta 44 soundcard has become the *de-facto* standard for A/D conversion for use with SDRs, there are a number of advantages to making your own ADC. These include having complete control of any software drivers needed to communicate with the ADC chips, as well as the optimisation of sampling rates and bit depths for individual signals.

It is also now possible to develop an ADC board which approaches the performance of professional sound cards - which is where the Janus board comes in. One great side effect of the consumer demand for high-quality PC sound cards has been the availability of a number of very high-performance, and low-cost, ADC and DAC converter chips that are ideal candidates for this project.

During the initial development of the Janus board, the Wolfson WM8785, Texas Instruments PCM4202 and Cirrus Logic CS5381 A/D chips were evaluated for use as the prime I and Q ADC converter. Whilst all three chips gave very high performance at 48k and 96ksps (kilosamples per second), it was found the noise floor beyond about ±60kHz increased substantially when operating at 192ksps. As a result, after a suggestion from Bob McGwier, N4HY, an AKM AK5394A was then evaluated and chosen for the prime ADC converter.

The Janus module has been interfaced to the open source *PowerSDR*^{TM} software by Bill, KD5TFD and enables users to select I/Q sampling rates of 48k, 96k and 192kbps at 24-bits. VK6APH's development board for Janus is shown in **Fig 8.18**.

The Texas Instruments TLV320AIC23B was used as the ADC for microphone and line inputs, and D/A converters for audio out and I/Q signals for the transmitter. This remarkable chip contains a microphone amplifier, with bias feed for an electret microphone, stereo line in and out and has stereo 16-bit A/D and D/A converters, together with a 35mW stereo headphone amplifier. Originally, the TLV320AIC23B was designed for use in MP3-format music players and the one-off price for this chip is U$7 - we have a lot to thank MP3 design engineers for!

Since reserving lines on the Atlas backplane at such an early stage is fraught with future issues, the Janus module incorporates a Complex Programmable Logic Device (CPLD) to provide complete flexibility in pin assignments.

The purpose and operation of the Janus board- and the practical reasons for it - will be described in more detail later in this chapter.

Ozymandias - Connecting to the World

Ozymandias (Ozy) is a Field Programmable Gate Array (FPGA)-based interface controller card that provides the input and output connections to the real world, so the HPSDR can be easily controlled by a personal computer and its operator.

Firstly, an Altera Cyclone II FPGA (EP2C5Q208C8) is used to provide numerous control lines for interfacing the various boards connected to the Atlas backplane. The Cyclone II contains over 4,600 logic elements and some 26 4k RAM blocks plus almost 120,000 RAM bits.

The Ozy module also provides a high-speed USB 2.0 interface to the controlling PC. The USB interface uses a Cypress FX2 chip, which greatly simplified the design and development effort necessary for the interface. The FX2 supports full-duplex USB communications at a measured 35MB/s - enough speed to allow sufficient bandwidth for future requirements.

The FPGA also provides the necessary control logic and data formatting for the Janus board, as well as serial and parallel interfaces for user-defined I/O. The software interface to the Janus module has been written in *Verilog* by VK6APH and KD5TFD.

One highly useful feature of the Ozy board is its potential ability to measure the various high-frequency crystal oscillators used by the HPSDR through the use of a 10kHz or 1Hz clock from a Global Positioning System (GPS) receiver. A GPS clock could be used to 'gate' the various oscillators and report the count to the operator's personal computer via the USB interface. Since the personal computer will have prior knowledge of exactly what these frequencies should be, then any errors can be corrected in software. The result of this would be that the HPSDR will be a radio with extremely high frequency accuracy and stability - a boon for microwave and earth-moon-earth operators.

Discussions on the HPSDR reflector have indicated there is interest in designing a GPS-locked reference for the project on its own Atlas-compatible board. This board will be named 'Gibraltar'! The project leader for Ozy was Phil Covington, N8VB. Ozy and its purpose and operation - and the practical reasons

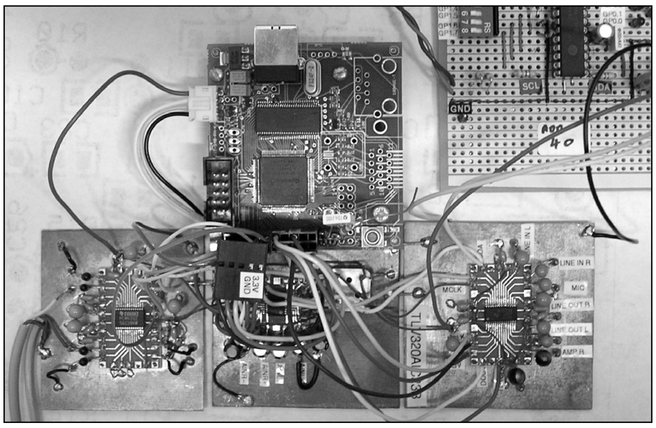


Fig 8.18: VK6APH's development board for the Janus DAC/ ADC

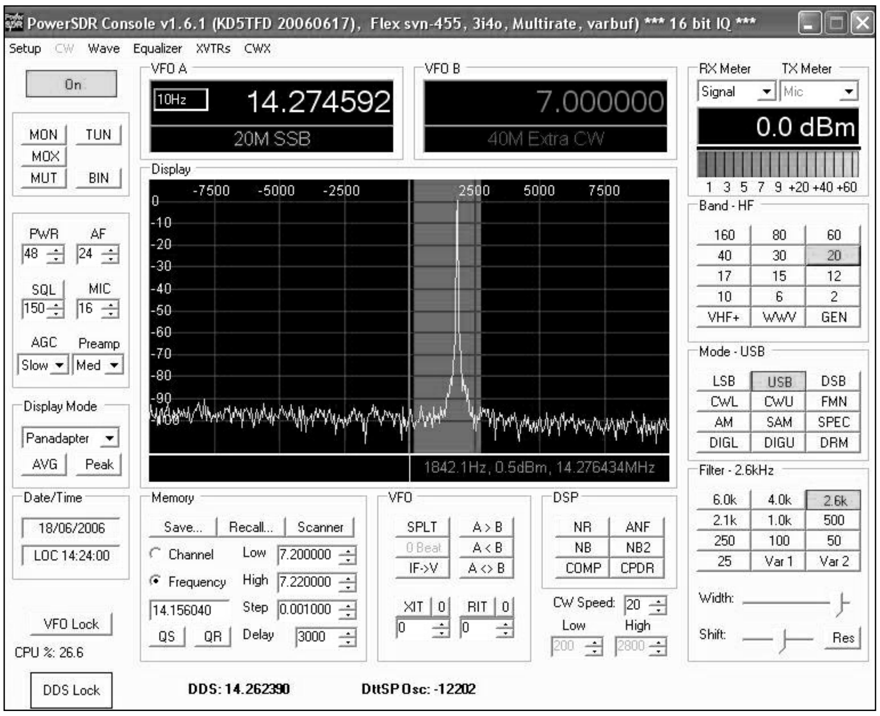

PowerSDR Console v1.6.1 (KD5TFD 20060617), Flex svn-455, 3i4o, Multirate, varbuf) *** 16 bit IQ ***
Setup CW Wave Equalizer XVTRs CWX
On
MON TUN MOX MUT BIN
PWR 48 AF 24
SQL 150 MIC 16
AGC Slow Preamp Med
Display Mode Panadapter AVG Peak
Date/Time 18/06/2006 LOC 14:24:00
VFO Lock
CPU %: 26.6
DDS Lock
VFO A 10Hz 14.274592 20M SSB
VFO B 7.000000 40M Extra CW
Display
1842.1Hz, 0.5dBm, 14.276434MHz
RX Meter Signal TX Meter Mic 0.0 dBm
Band - HF 160 80 60 40 30 20 17 15 12 10 6 2 VHF+ WWV GEN
Mode - USB LSB USB DSB CWL CWU FMN AM SAM SPEC DIGL DIGU DRM
Filter - 2.6kHz 6.0k 4.0k 2.6k 2.1k 1.0k 500 250 100 50 25 Var 1 Var 2
Width: Shift: Res
Memory Save... Recall... Scanner Channel Frequency Low 7.200000 High 7.220000 Step 0.001000 Delay 3000 14.156040 QS QR
VFO SPLT A > B 0 Beat A < B IF->V A <> B XIT 0 RIT 0
DSP NR ANF NB NB2 COMP CPDR CW Speed: 20 Low 200 High 2800
DDS: 14.262390
DttSP Osc: -12202

Fig 8.19: Screen shot of Mercury digital down conversion receiver running into *PowerSDR™* software

for a board of this kind - will be described in more detail later in this chapter in the section on PC to SDR communications.

Mercury Receiver - Analogue to Digital Conversion at the Antenna

Perhaps the most exciting of all the HPSDR modules, the Mercury board enables direct sampling/reception of the 0 - 55MHz radio spectrum, effectively at the antenna.

Based on a Linear Technology LTC2208 130Msps 16-bit ADC, the board contains its own FPGA to undertake 'Direct Down Conversion' (DDC) to approximately 200ksps for transfer over the Atlas bus to the USB interface on the Ozy board. Mercury down-samples' in its own Altera Cyclone II FPGA, not unlike Mark Ettus's commercial USRP SDR [38].

The Mercury sampling converter/receiver covers the range 0 - 55MHz when fundamental sampling is used - and well into the UHF range on harmonic sampling. What is even better is Mercury has a Blocking Dynamic Range (BDR) of 119dB. The BDR was measured at 100kHz and 5kHz for 1dB gain compression with similar results. Even the original Mercury prototype had a dynamic range of about 100dB. A screen shot of the Mercury prototype running into the *PowerSDR*™ software (**Fig 8.19**) illustrates this point.

The LT2208 was connected via a Xylo FPGA board [39] over USB 2 to *PowerSDR*™, with an input signal level of 0dBm. Special thanks must go to KD5TFD for modifying *PowerSDR*™ to take the 16-bit data from the LT2208. A Linear Technology LTC 2208 evaluation board was used in early trials of the Mercury principles.

One huge - and often unrealized advantage of the direct sampling approach is that no I/Q phase or amplitude balancing is required, since these signals are generated from the incoming signal and are created in software. This neatly gets rid of the age-old problem of image rejection associated with direct conversion receivers and latterly with down-conversion SDRs such as the Flex-Radio SDR-1000 and the SoftRock series. In the case of direct sampling and the Mercury module, there is simply no image to get rid of.

The Mercury board will be discussed in more detail later in this chapter, in the section on Digital Down Conversion (DDC). It is currently available from the TAPR as a complete and assembled printed circuit board.

Penelope - Companion Exciter for Mercury

Penelope is a half-watt transmitter/exciter board, intended as a companion to the Mercury HF receiver board.

The Atlas bus-compatible Penelope transmitter uses Digital Up Conversion (DUC) techniques and processes the I and Q signal from a PC (or a future HPSDR Sasquatch DSP board) directly without the need for a sound card. I and Q balancing is not needed in Penelope, due to the RF waveform being digitally generated.

The DUC is FPGA-based, so as to enable future software upgrades and has a USB interface to a PC via the Ozy board.

Some of the features of Penelope include:

- 1.8 - 55MHz frequency coverage.
- 0.5W PEP output.
- Low level output for VHF/UHF transverters.
- Capable of SSB, AM, C-AM, FM, CW, PSK etc.
- RF phase and magnitude outputs for future 'Envelope Elimination and Restoration' (EER) power amplifier
- Open drain FET for Press-To-Talk (PTT) control of external amplifiers.
- Seven open collector outputs for the control of external devices (eg a linear power amplifier).
- Optional on-board microphone ADC for use without a Janus card
- An ADC is used for Automatic Level Control (ALC) or PA linearization, etc.
- The ALC is processed in the FPGA so as to avoid delays associated with PC processing.

In addition, the Penelope board has a number of different options for controlling its frequency, to suit different forms of frequency standard. These are:

- On-board high performance 125MHz crystal oscillator.
- External 125MHz source.
- On-board oscillator can be phase locked to 10MHz reference (e.g. HPSDR Gibraltar board).
- On-board 10MHz OCXO/TCXO option.

Alexaires - Receiver BPF / Transmitter LPF

Alexaires - 'Alex' for short - is a combination RF receiver preselector and transmitter low pass filter bank for use with the HPSDR's Mercury receiver and Penelope transmitter, and optionally, with an associated RF power amplifier up to 100 watts peak. It has been designed by Graham, KE9H, and Phil, VK6APH.

As a receiver preselector, the purpose of Alex is to reduce the level of out-of-band signals at the input of Mercury and, importantly, to suppress any signals at the sampling image or alias frequencies. As a transmitter low pass filter, Alex will suppress the harmonic energy typically generated by an RF power amplifier, as well as the images or aliases that appear at the sampling clock frequency (122.8MHz) plus/minus the operating frequency. The transmit low pass filters are also used for additional Mercury receiver input band limiting.

Normally one of the low pass filters on the Alex transmit board will be paired-up with one of the high pass filters on the Alex receiver board. There is an additional 33 or 55MHz low pass filter on the receiver board, plus the 6-metre transmit low pass filter is in-line at all times to help suppress VHF images.

At the time of writing, Alex was undergoing beta testing and is expected to be available as two complete assembled PCBs from the Tucson Amateur Packet Radio Group (TAPR).

Pennywhistle 20W RF Power Amplifier

Pennywhistle is a compact RF power amplifier stage that can be used with the Penelope transmitter and the Alexaires filters to make a complete 16 to 20 Watt transmitter. This inexpensive amplifier can quickly be used to get an HPSDR on the air - either 'bare-foot' or as a driver for a larger HF linear amplifier. It covers the long-standing amateur bands between 1.8 and 54MHz - the same as the rest of the HPSDR transmit/receive boards - and measures 10 cm by 8 cm (half Euro-board size.)

The output stage of Pennywhistle is a single stage amplifier that uses a pair of TO-220 16-Watt Mitsubishi RD16HFF1 devices in push-pull. The amplifier has approximately 19dB gain, depending where it is biased, and will deliver 16 to 20 watts output with 0.25W of drive.

At the time of writing, an alpha version of Pennywhistle was being tested by its designer, Graham, KE9H, Phil, VK6APH, Bill KD5TFD, and Dick, K9IVB. It is anticipated it will be sold as a bare printed circuit board 'build-it-yourself' kit by the TAPR.

LPU - a Linear Power Unit

The 'LPU' is a simple HPSDR power supply and provides a convenient low-noise solution to power the Mercury receiver, Penelope transmitter, Ozy PC communications board and Alexaires filter board from a 12V supply until the more complex Demeter power supply is completed, in order to make a basic and functional HPSDR transceiver.

Demeter has the following specifications:

Input power:	12.5V - 14.5V regulated DC input
Outputs:	+12V@2A, +5V@2A, +3.3V@2A (optional), -12V@100mA

There is a poly fuse on each regulator input: +12V, +5V, +3.3V (optional) and -12V.

The LPU plugs into the Atlas backplane ATX connector and can be directly mounted on the backplane. Other features include a strap disable for -12V to reduce noise, dual power-pole input connector and an internal dual power-pole output connector for a power amplifier or other accessories. At the time of writing, the LPU was undergoing alpha testing by its designer, Scotty WA2DFI and is expected to be available as a kit from the TAPR.

Sasquatch - DSP Back-end for 'PC-less' Operation

The Sasquatch board is a hardware digital signal processor-type 'back-end' for the HPSDR, intended for use by constructors who would like to operate the HPSDR as a stand-alone radio rather than as an attachment to a PC. The board may even allow real knobs and buttons to control the radio, rather than the software keyboard/mouse controls of a PC-based SDR. The project leader for the board is Lyle KK7P.

The board, which is presently in the planning phase, is anticipated to have the following features:

- Texas Instruments TMS320C6726 32-bit Floating Point Digital Signal Processor.
- Its own Field Programmable Gate Array (FPGA).
- A Flash memory for self-booting / starting.
- Connector for a JTAG-based emulator.
- Analogue and digital Input / Output.
- Power consumption of less than one watt.

Other HPSDR Modules

Other modules are being developed by various project leaders. These include a high efficiency HF power amplifier (Thor), an enclosure to contain the HPSDR boards (Pandora), the Demeter high-performance power supply and a general purpose input/output board that sits on the Atlas bus (Epimetheus).

There is still much to be done in bringing the HPSDR to fruition. For those experimentally-minded radio amateurs involved in the project, this may turn out to be the golden age of (software defined) radio. If you have an interest in 'bleeding edge' technology, surf to the HPSDR website [8] and reflector and sign-up- it should be a fun ride.

µWSDR VHF/UHF/MICROWAVE PROJECT

Like the High Performance Software Defined Radio project, the µWSDR is a GNU-based next generation SDR. However, while HPSDR is mainly aimed at 0-55MHz, the µWSDR - or 'microwave SDR' is squarely targeted at microwaves and VHF/UHF. GNU is a free UNIX-like operating system. Variants of GNU, which use the Linux kernel, are now widely used. As in the HPSDR, one of these GNU variants will run the µWSDR in addition to Windows™.

The µWSDR is being designed and developed by a group of microwave enthusiasts, which includes Chris Bartram, GW4DGU, and Grant Hodgson, G8UBN, who are working on the RF design for the project. Other key members of the team are Tobias Weber, DG3YEV (firmware and digital hardware design), Jonathan Naylor, ON/G4KLX (PC software and web pages), David Wrigley (PIC firmware for the front-end boards) and Chris Bryant, G3WIE.

The European-based project has a dedicated web site [40] which explains the basic ideas behind the µWSDR.

Like the HPSDR, the rationale behind the µWSDR project is to break the overall design of a high performance SDR into two self-contained modules - a separate RF front-end for each band and a standardised digital 'back-end', which will build into a self contained unit (with the exception of the PC which carries out control and digital signal processing functions). Each of the RF front-end modules is being designed by an individual or small group and connects to the digital back-end using a standard connector.

The philosophy behind µWSDR is strongly 'open source' - the same as the HPSDR - so the circuit of each module, and any

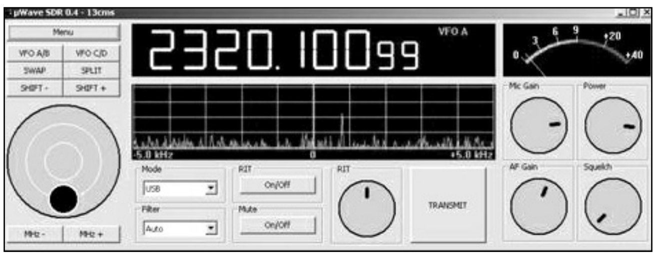


Fig 8.20: µWSDR control panel, as it will appear on a PC

software it uses, will be published on the Internet and freely available to anyone who wishes to use them. **Fig 8.20** shows the µWSDR control panel, as it could appear on a PC screen.

Neither the microwave SDR nor the HPSDR will use PC soundcards for analogue-to-digital or digital-to-analogue conversion, as does the Flex-Radio SDR-1000 or the SoftRock kits.

By now, some of you will probably be thinking that µWSDR and HPSDR appear to have an awful lot in common and it would seem a waste of resources if each group were to develop all of its own hardware, firmware and software, when the other group may have already done some suitable work. Luckily this thought has already occurred to members of both groups and discussions have already taken place so ideas, circuits and code can be exchanged freely.

There are a couple of key philosophical differences between the µWSDR and HPSDR designs.

Whilst the µWSDR project is aimed at attracting newcomers to try out the microwave bands and thus has a low-cost aim for the basic kit, HPSDR is aimed primarily for no-compromise high performance on the amateur bands between 1 and 54MHz, whatever the cost. That being said, the HPSDR team are confident their radio will cost much, much less than a current top-line HF radio, such as the Ten Tec Orion 2.

The basis of the µWSDR, in common with most SDRs, is to move many of the functions of the intermediate frequency and audio stages of a conventional analogue superheterodyne radio transceiver into software, but still use hardware to down-convert the operating frequency to a very low intermediate frequency (of around 24kHz) and then digitize the signal to enable the processing of it to be done on a personal computer.

A block diagram of the µWSDR signal path is shown in **Fig 8.21**. Each RF front-end module of µWSDR is anticipated to incorporate the local oscillator, receive RF amplifier and mixer, transmit mixer and associated amplifier. The two major design goals are to generate a 200mW transmit signal at the signal frequency and have a receiver with a noise figure that does not exceed 3dB.

By using a different RF front-end module for each desired band, only some minor reprogramming of the SDR's CPU (central processing unit) should be necessary to support each band. The first band module to be available will be the one for 13cm. In September 2008, the GeMMA transmit board was in the process of being tested.

The back-end of the µWSDR will contain high quality analogue-to-digital (for receive) and digital-to-analogue (for transmit) converters, the Central Processing Unit (CPU) and an Ethernet interface to the personal computer.

In contrast to the HPSDR, which uses a USB 2-type PC interface, 100Mbps Ethernet was chosen for this purpose on the µWSDR.

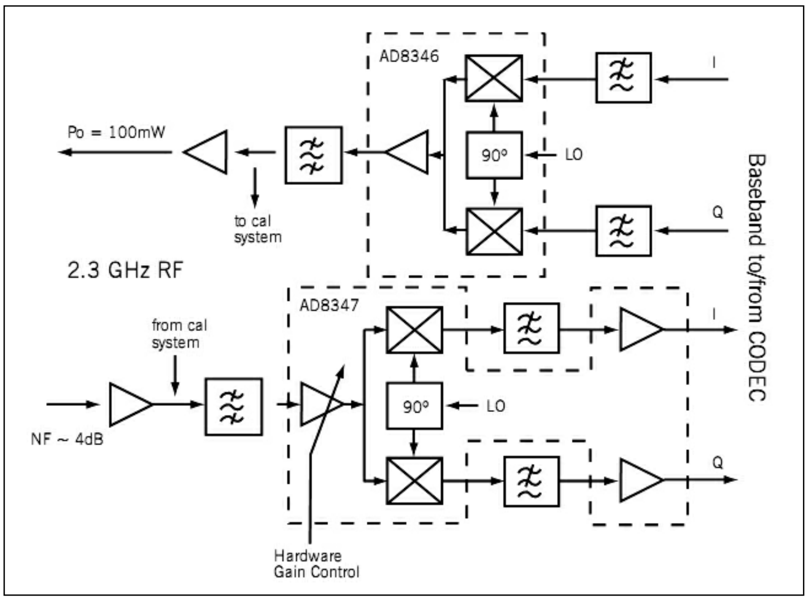


Fig 8.21: Block diagram of µWSDR signal path

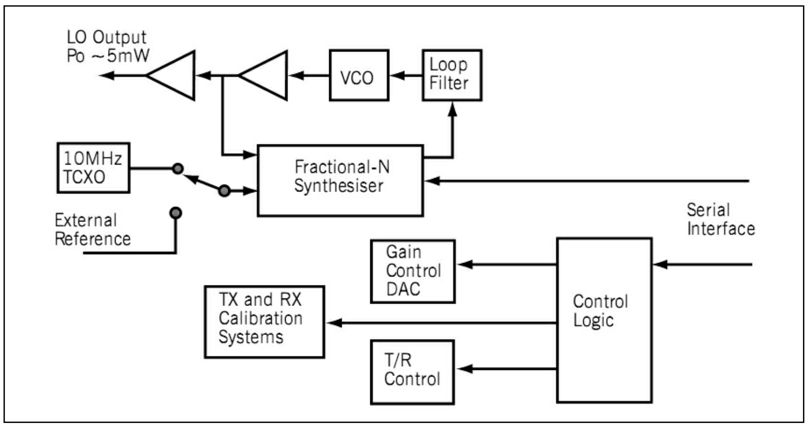


Fig 8.22: Block diagram of µWSDR local oscillator and control circuit

LOCAL OSCILLATORS FOR SDR

The Direct Digital Synthesis (DDS) local oscillators used to control the operating frequencies of most modern HF, VHF and UHF transceivers can provide very low phase noise, but can also produce numerous spurious outputs. Although there has been a considerable reduction in the level of spurs in the latest generation of DDS devices, they can still be significant when designing very high performance receivers.

An option often used to reduce spurs is to feed the output of the DDS into a 'clean-up' Phase Locked Loop (PLL). Whilst this can be effective in reducing the level and number of spurs, the phase noise of the overall local oscillator is degraded due to the oscillator used in the PLL.

A number of high performance receivers overcome this problem by dividing down a VHF oscillator that uses a DDS to provide fine tuning steps. Each time the oscillator frequency is divided by two, the phase noise is theoretically reduced by 6dB. Examples of receivers that use this technique are the TT Orion (which has an oscillator around 550MHz) and the CDG2000 [41].

Given a conventional SDR digital signal processing (DSP) back-end (eg a PC using a sound card as an ADC/DAC and running *PowerSDR*™, *Rocky* or *KGKSDR* application software), we can tune using the software a few kilohertz around the wanted frequency and can also tune by moving the sound card sampling rate around the local oscillator frequency). This relieves us from the need to 'step' the main local oscillator in fine steps - and instead this can 'tune' in say 10kHz steps, with the fine tuning being done in software.

This greatly simplifies the design of the local oscillator for an SDR. By using a microwave oscillator and dividing this down to the HF range, a substantial reduction in phase noise can be achieved. In the proposed Horton module for the HPSDR, The intention is to use a 'canned' 2.4GHz VCO that 'tunes' in 800kHz steps. This is subsequently divided to produce I and Q inputs to the mixer stages.

The µWSDR team favour the use of fractional-N phase lock loops for the local oscillator - technology that is likely to be familiar to microwave enthusiasts. For the lower frequency bands, the µWSDR oscillator will also be a microwave fractional- N synthesizer, followed by a divider. A block diagram of the µWSDR local oscillator and control circuit is shown in **Fig 8.23**.

The µWSDR team is looking to tune the local oscillator in 10kHz steps, which simplifies its design. Provided the bandwidth available at the intermediate frequency is wide enough, there is no need for the local oscillator to tune in small steps; the gaps in between are filled by a software 'oscillator' running within the PC.

ADDING AN SDR BANDSCOPE TO YOUR ANALOGUE HF TRANSCEIVER

DXers or contesters who have experimented with SoftRock hardware and *Rocky* or *KGKSDR* software often wish their analogue HF transceivers had the amazingly sensitive bandscope facilities that the combination of the SoftRock/*Rocky* [1] or *KGKSDR* [3], their personal computer and a 'prosumer' grade sound card provide.

One option is to go and buy a software defined radio transceiver that will run software of this kind. However, a Flex-Radio SDR-1000 or FLEX-5000 are not cheap and are a big leap of faith for a radio amateur who is more operator than experimenter.

The good news is there is a middle way available. This involves 'tapping' one of the intermediate frequency (IF) chains in the receive section (see **Fig 8.23**) of a commercial HF transceiver and feeding this output into a SoftRock receiver, so as to provide digital signal processing and a display using *Rocky* or *KGKSDR*, and digital-to-analogue conversion using an M-Audio Delta 44 soundcard.

The majority of the HF transceivers popular over the last decade - such as the Yaesu FT-1000 series or the Kenwood TS-850 - have a first IF at VHF and a second IF between 8 and 9MHz; 8.215MHz in the case of the FT-1000 and 8.83MHz in the case of the TS-850. If this second IF output is fed into a SoftRock that is fitted with a suitable crystal and a bandpass filter, then the SoftRock can be used to provide a separate receiver of its own. This second receiver can provide a bandscope, digital filtering and high-quality digitally-processed audio output (if fed into a soundcard like the Delta 44).

In this application, the wide bandpass filters and first IF roofing filters (which in the case of the latter may have a bandwidth of up to 10 to 15kHz) typically used in this architecture of radio are an advantage, ensuring that the bandscope facilities provided by the SoftRock/transceiver/software combination are sensitive across a wide chunk of frequency spectrum.

Using a SoftRock in this manner was pioneered in 2005 by Alex Shovkoplyas, VE3NEA, the author of *Rocky*, who used a SoftRock as a digital back-end to his Kenwood TS-570S.

A few years ago, KB9YIG built up a v6 SoftRock for VK6VZ to experiment with using his Yaesu FT-1000 as an IF signal processor (connected before the 8.215MHz 2nd IF crystal filter). VK6VZ is a very keen 160m weak signal CW DX operator and the idea was to have a very sensitive bandscope that could 'see' weak signals appearing across the CW portion of 160m at sunset/sunrise times. It would also enable him to compare the FT-1000's famous analogue receiver (with 500Hz crystal filters in both the second and third IFs, plus a brilliant audio peaking filter and IF shift and width controls) with the Softrock back-end, which was to use *Rocky* and *KGKSDR* software that have digital filtering variable down to several tens of Hz.

For the FT-1000's 8.215MHz IF application, KB9YIG used an 11.0MHz crystal, used with 1/3 sub-harmonic sampling. The centre frequency is about 3* (11.0 - 0.003) / 4 = 8.248MHz, ie 33kHz above the IF frequency. The modified SoftRock v6 was fed into a M-Audio Delta 44 soundcard (for analogue-to-digital conversion) that can sample at 96kHz, so the IF tuning with the SoftRock is from 8.2MHz to 8.296MHz. This gave excellent coverage of the FT-1000D 2nd IF passband. The M-Audio Delta 44 soundcard ran from a Pentium IV personal computer.

KB9YIG's completed SoftRock v6 was mounted in a diecast aluminium box by VK6VZ and then VK6APH worked out how to interface it to the FT-1000D and modified the latter. VK6APH had to drill a hole in the rear of the case of the FT-1000, in order to get the second IF connection out via a new RCA socket.

In essence, VK6APH got the IF output from the FT-1000 by 'tapping' an IF buffer amp (Q2003) at a low impedance point (one of the gates, to be precise) with a 22pF capacitor, following this with a 10k/47ohm attenuator - see Fig 8.27.

Note that the surgery necessary to the FT-1000 in order to get the second IF 'tap' was difficult to perform and required some dismantling of the radio, a cool head, superb soldering skills and the use of a very fine point soldering iron. It is potentially a very good way to break your beloved/expensive FT-1000.

To power the SoftRock, the 13.8V available at an RCA socket on the rear of the FT-1000 was used.

With the antenna socket of the Softrock connected to the new FT-1000 IF output, the SoftRock connected to the PC's sound card in the usual manner, a resonant antenna connected to the FT-1000 and the latter tuned to the centre of the 7MHz band, this level of attenuation gave a noise floor on the KGKSDR bandscope of about 10dB above the bandscope's base line noise when there was no antenna connected to the FT-1000.

The 10 kilohm variable resistor was set between one third and a half of its value, in order to provide the correct level signal into the SoftRock. This adjustment isn't critical but care should be taken so as not to overload the SoftRock/soundcard combination.

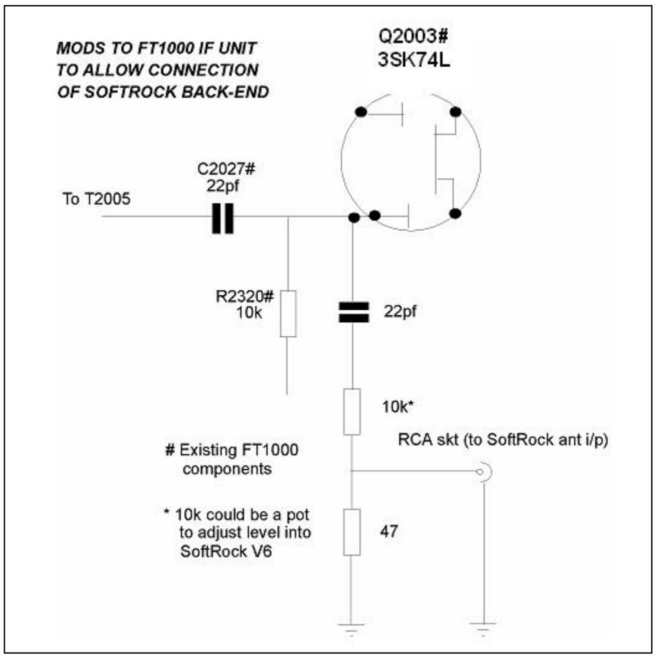


Fig 8.23: Modifications to the Yaesu FT-1000D to allow connection of a SoftRock v6 'back-end'

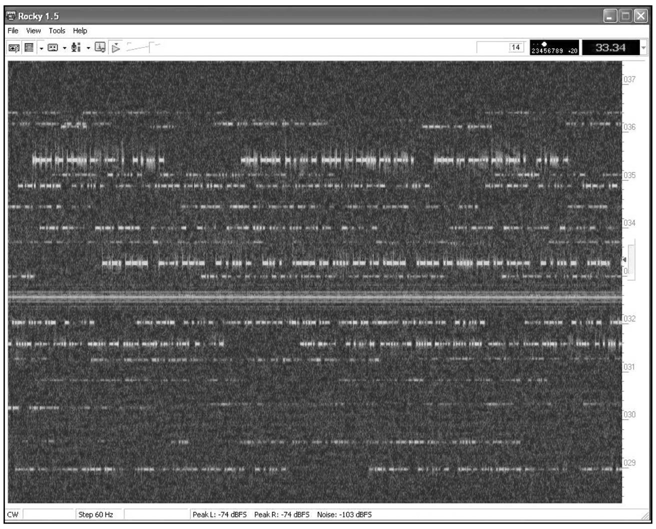


Fig 8.24: Activity during a 7MHz CW contest during October 2006 viewed using the *Rocky* waterfall display function. The small arrow head on the right hand side of the display shows the frequency to which the FT-1000D/SoftRock v6/*Rocky* 1.5 combo are tuned.

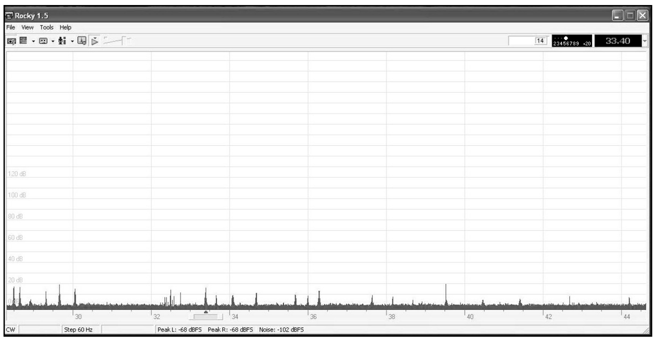


Fig 8.25: Activity during a 7MHz CW contest during October 2006 viewed using the *Rocky* bandscope display function

The adjustment is easiest carried out by using the *Rocky* software and a signal generator. Making sure that the FT-1000 noise blanker is switched off, a signal of around S7 (on the FT-1000 'S' meter) was injected into the FT-1000 antenna input and the variable resistor adjusted until the point that the level of the signal shown on the *Rocky* bandscope no longer increased.

Next, the FT-1000 noise blanker was switched on and the signal level checked to make sure it was the same as before on the *Rocky* bandscope. If not, it may be necessary to adjust the pot slightly until this situation is achieved (ie that the signal level on the *Rocky* bandscope looks the same with the noise blanker switched on or off).

The SoftRock v6 can be overloaded with signal from the FT1000, so care needs to be taken to adjust the gain of the SoftRock in *KGKSDR* and *Rocky* so that this does not happen.

Results

The results were terrific - VK6APH and VK6VZ could see ±30kHz from the frequency that the FT-1000 was tuned to, using *Rocky* or *KGKSDR* as our bandscope. Given the narrow nature of the section of 160m where CW operation takes place (1800 to 1835kHz) this enabled us easily to see all of this part of the band in one view.

The SoftRock back-end also gives a useful second receiver (strictly speaking, a third) for the FT-1000. You can see on the *Rocky* or *KGKSDR* bandscope where the main FT-1000D receiver is tuned, since the 8.215MHz crystal filter leaves a 'suck out' in the passband (ie in the form of a depression in the level of the trace) on the *Rocky* and *KGKSDR* bandscopes.

In *Rocky*, we set the local oscillator (LO) frequency to 0 in the Settings dialog, while in *KGKSDR* we had to set the LO frequency to 0.001kHz because it wouldn't accept a 0kHz LO. We also found that in *KGKSDR*, the bit depth needed to be set to 16-bits for acceptable performance.

After experimenting with the FT-1000 / SoftRock combination for a few weeks, VK6VZ found the best way to use it for CW was with the *Rocky* software in waterfall bandscope mode - see **Figs 8.24 and 8.25**. The screen was set-up so that the IF offset of the SoftRock (in practice with FT-1000 on CW, about 33.34kHz) was in the centre of the screen. This means that the SoftRock receiver is essentially tuned to the same frequency as the FT-1000 receiver, so you can transceive, using either receiver for reception.

The comparison showed that weak CW signals were, on balance, about equally readable on both the FT-1000 and SoftRock receivers. However, the *Rocky* bandscope (particularly in waterfall mode) could clearly see signals that were inaudible or barely audible on the FT-1000.

On one amazing occasion, VK6VZ watched K3NA appear slowly from the noise at VK6 sunrise on 1.8MHz on the *Rocky* waterfall, actually reading characters from his callsign on the screen several minutes before his signal actually became audible. If VK6VZ hadn't been able to 'see' K3NA's signal, he would probably given up listening.

At the same time whilst listening for K3NA's signal, VK6VZ could watch the rest of the CW band on the waterfall display, to see what other weak signals came up.

When it came to SSB operation, signals were much more readable/pleasant to listen to from the SoftRock than from the FT-1000. VK6VZ's preferred software for SSB reception was *KGKSDR*, with its excellent AGC characteristics for SSB. VK6VZ's observation was that good SSB signals sounded awesome and poor ones sound poor on the FT-1000/SoftRock/*KGKSDR* combo - in comparison, on the FT-1000 alone, they all sounded similar and relatively muddy.

This performance isn't just down to the direct conversion SoftRock receiver and *KGKSDR* software, but to the excellent analogue-to-digital conversion of the Delta 44 soundcard and the hi-fi audio it provides to a pair of PC speakers.

The main thing to understand here is that the bandscope you get from *Rocky*/*KGKSDR*, etc is able to 'see' signals that you can hardly hear (and in some cases, not hear). In comparison, apparently the bandscopes used in the current generation of HF transceivers are fine at seeing strong signals, but not necessarily very weak ones.

The only downside VK6VZ has found is that if you use the SoftRock as the main receiver, it acts as a monitor receiver on transmit and the audio coming out is delayed by a few hundred milliseconds - which is terribly distracting when you are speaking or sending CW!

However, it is a relatively simple process to arrange for the SoftRock to be muted on transmit - KB9YIG has provided two terminals for this facility on the v6.

For those who are interested in finding out how to modify their particular HF transceivers to work with a SoftRock in this manner, the SoftRock 40 web pages [13] contain a section devoted to 'application notes' on a variety of radios [42]. If you cannot find your particular model of radio, join the SoftRock 40 reflector and you can e-mail its members to find out if any of them have devised modifications for it.

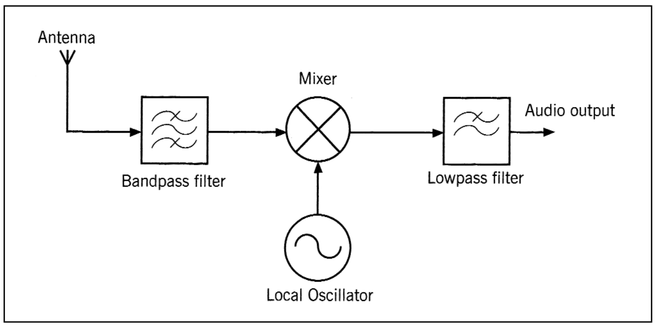


Fig 8.26: Direct conversion receiver

WHY SDRS NEED I AND Q SIGNALS

Most of the Software Defined Radios (SDRs) we have talked about so far - from the Flex Radio SDR-1000 to the various SoftRock receivers to the G3PLX Zero IF method, have had one thing in common - they all process two signals, consisting of an I (for in-phase) and a Q (for quadrature) signal.

Why do we need these two signals when perhaps at first consideration we only need one? Understanding this idea is the key to truly understanding the current generation of SDRs.

Fig 8.26 shows the block diagram of the front-end of a direct conversion receiver. The signals picked up by its antenna are first filtered to ensure so that only those within the frequency band of interest is passed and then these signals are applied to a mixer. The other signal 'input' to the mixer is a local oscillator which, in the case of a CW signal, is set to be a few 100Hz away from the wanted signal, in order to give us the audible beat note required to decipher the signal.

The output of the mixer is filtered to preserve the difference between the wanted signal and the local oscillator signal. Since this difference is within the audio range, the resulting audio frequency signal is passed on to our PC sound card for processing.

With the SDRs we have considered so far, they have what seems to be two identical receiver chains - see Fig 8.27, which shows a block diagram of the classic SDR. As you can see, we have a second mixer which takes the same filtered antenna signal as the first, as well as having an identical local oscillator frequency feeding into it and an identical filter on its output. The output from this second receiver chain apparently provides the same audio signal output as the first.

So these two output signals are the same - right? Well, not quite - there is actually an important difference between them. If you look closely at Fig 8.31, you will see that the local oscillator connection to the second mixer is passed through a block that is marked '90 degree phase shift'.

This means that whilst the *frequency* of the local oscillator fed to the second mixer is exactly the same as the first, the *phase* has been shifted by 90°. This 90° phase shift - or difference will also be present on any signal that passes through the second mixer, hence the audio output signal from the second mixer will be at 90° to that being produced by the first mixer. Because of the phase difference between the two local oscillator signals, the I and Q signals have a 90° phase difference.

If just the Q signal is fed through a band pass filter, note that this causes an additional 90 degrees phase shift for all signals passing through the filter. The filter bandwidth depends on the mode being used; for CW this filter could be, say, 500Hz and for SSB, say, 3kHz.

The I signal goes though a similar bandwidth filter but does *not* get the additional phase shift. So now the I and Q signals have a 180° phase shift and if we add them, we get double the signal out and if we subtract them we get zero.

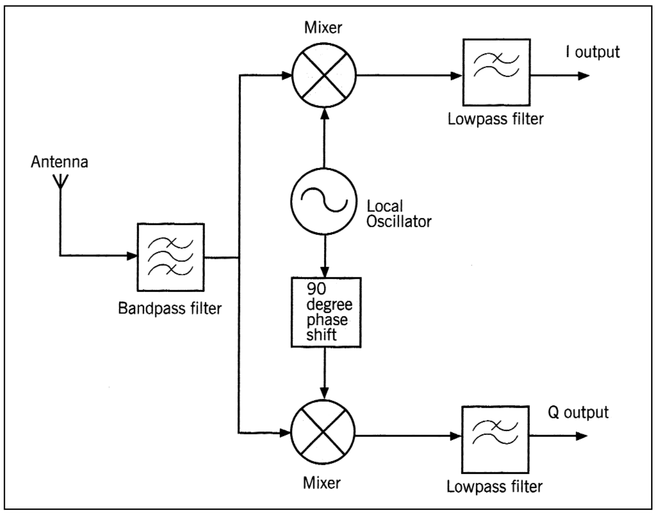


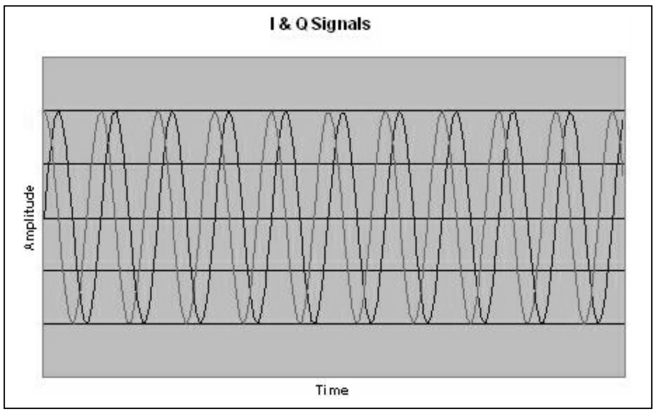
Fig 8.27: SDR front-end

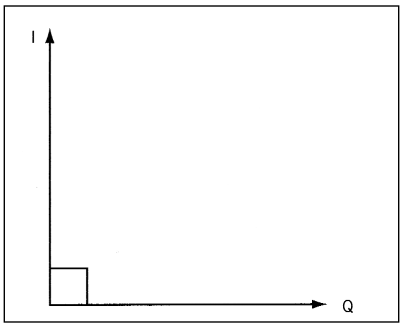
This is the key point in understanding how I and Q demodulation works - as we tune across a signal from, say, 1kHz above the signal to 1kHz below the signal, then as we pass through zero-beat the phase relationship between the I and Q signal changes by 180°.

So let's say that the above zero-beat I leads Q by 90°, the below zero-beat I will lag Q by 90°. If you have a dual channel oscilloscope and look at the I and Q outputs of a SoftRock SDR and tune a signal generator through zero-beat, you will see this effect.

What this means is that a signal above the local oscillator frequency is received and any which are below the local oscillator frequency are not received, ie we have an USB (upper sideband) receiver. Similarly, if I and Q are subtracted rather than added at the output, then the reverse applies - we get an LSB receiver.

If a dual channel oscilloscope is connected to the output of the two mixers, we would see a trace similar to that shown in Fig 8.28. We can also represent the two signals as vectors with equal amplitudes and an angle of 90° between them - see Fig 8.29.





(above) Fig 8.28: I and Q signals at the output of the mixers

(left) Fig 8.29: I and Q as vectors

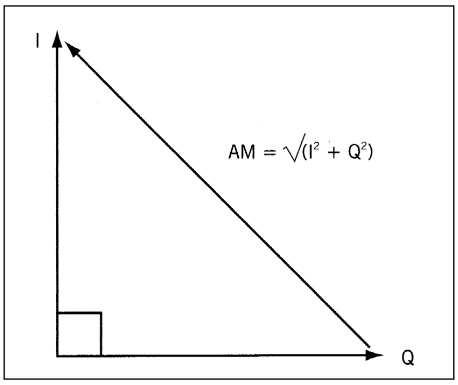


Fig 8.30: AM demodulation using I and Q

The vectors are both rotating at the difference in frequency between the input (signal) frequency to the mixers and the local oscillator frequency, but Fig 8.33 shows a 'snap shot' at a moment in time, so they appear stationary. The vector representation of the I/Q signals is very useful for understanding some additional theory, as we will see in a moment.

The two audio signals are said to be 'in quadrature', or 'phase quadrature', and are labelled 'I' (for in-phase) and 'Q' (for quadrature). The I signal by convention is the one that reaches its most positive value first. You will also see this I and Q signal pair referred to as a complex signal (though hopefully not complicated after the preceding explanation!).

In order to get a complete understanding of what is going on, it is necessary to get mathematical for a moment. The basis for DSP processing of a modulated signal is the use of complex exponential representations, instead of sines and cosines.

Now, the cosine signal is simply the 'real' part of the complex exponential, while the sine signal is the 'imaginary' part of the complex signal, as per Euler's formula:

exp^jt = cos(ωt) +j sin(ωt)

If you consider the real and imaginary parts of the complex signal as components of a vector that rotates around a point, you will see that if ω is positive, the vector rotates counterclockwise, while if ω is negative, the vector rotates clockwise.

If the local oscillator signal is also considered as a complex signal, multiplying the complex input by the complex local oscillator produces only a sum frequency - there is no difference frequency. In mathematical terms, when multiplying two exponentials, you simply add the exponents - which simplifies processing:

Z = exp(j*f1) * exp(j*f2) = exp(j*(f1+f2))
= cos(f1 + f2) + j sin(f1 + f2)

So, back to our original question: "why go to all the trouble and expense to generate two signals that are 90° out of phase with each other?"

Someone once said "give me I/Q and I can demodulate anything" [43]. And that, quite literally, is the reason we go to all this trouble. If you have I and Q signals then you can demodulate any signal - be it AM, FM, SSB, CW, PSK31, etc, or any new modulation system that anyone may dream up in future. Similarly, in the case of a transmitting a signal, if we generate the appropriate I and Q signals we can transmit any form of modulation.

AM Demodulation

Let us now look at some modulation systems and see how they can be demodulated if we have I and Q signals available. Let's start with a simple system - good old fashioned Amplitude Modulation (AM).

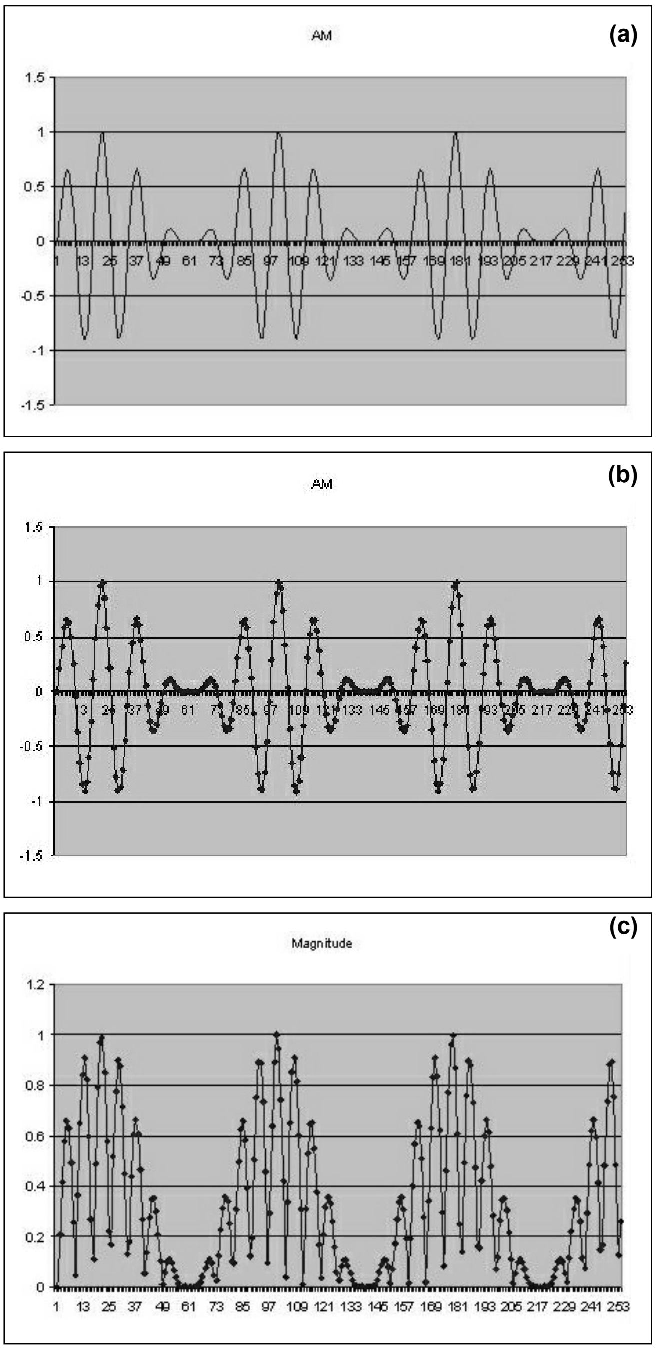


Fig 8.31: (a) AM signal at the input to the sound card; (b) Sampled AM signal; (c) Magnitude of AM signal

To demodulate an AM signal, we simply use Pythagoras' famous theorem about triangles (the square on the hypotenuse - the longest side of a triangle - is equal to the sum of the square of the other two sides) and take the square root of the sum of I squared plus Q squared, as per **Fig 8.30**.

$AM = \sqrt{(I^2 + Q^2)}$

Each of the I and Q vectors will be varying in amplitude in sympathy with the amplitude modulation on the incoming signal. Hence the hypotenuse of the triangle that we have just formed from the I and Q vectors will also vary in sympathy with the modulation.

This mathematical process is really easy to do in software and the following is the actual line of code inside a module of code called am_demod.c (from the open source *PowerSDR*^{TM} code and contained within *DttSP*, see [44])

```
am->lock.curr = Cmag (CXBdata (am->ibuf, i));
```

At this stage you may be thinking "hang on a minute, if both I and Q are amplitude modulated in their own right, then why not just take one of these signals and measure its magnitude directly, since this would give us the original modulation back?" This is rather like using a diode to demodulate AM as we do in an analogue receiver.

That's a very good question and the answer is "yes, you could." However, there is a good reason for not simply doing it that way but using both the I and Q signals instead, which is as follows.

Let's say we have mixed the incoming signal down to a frequency within the input range of a PC sound card, generally in the range 10 to 20kHz.

We will assume our signal has been mixed down to 10kHz and is being amplitude modulated with a 1kHz sine wave. As a result, at the input of the sound card we have a signal that looks like the one in **Fig 8.31(a)**.

Inside the sound card we convert the signal to a series of samples represented by the dots in **Fig 8.31(b)**. We then calculate the magnitude of each sample, which results in the signal appearing in **Fig 8.31(c)**.

If we were to feed this signal directly to the D/A converter in our sound card, we would end up with a strong component at 10kHz. We could pass the signal through a low pass filter before passing it to the sound card, but let's see what happens if we use the I and Q signals as proposed above.

Fig 8.32(a) shows the I and Q signals being fed into the sound card - note the 10kHz carriers are 90° out of phase with each other. As before, we then sample each signal inside the sound card, which results in the values shown with black or grey dots in **Fig 8.32(b)**.

We now apply Pythagoras theorem and the resulting signal is shown in **Fig 8.32(c)**. Notice that the 10kHz carrier component is no longer present in the output signal. By using the I and Q signals, we have eliminated the need to filter the demodulated signal.

What is even more useful is what would happen if we had mixed the AM signal down to an even lower IF, say 1kHz. Since this is within the audio range of a typical AM signal, it would have been impossible to filter out of our demodulated signal. However, when using the above I and Q technique, the IF frequency does not appear in the demodulated output. Note that the technique works even if we use zero Hz as the IF.

CW and SSB Demodulation

Now let's look at how we demodulate a CW or SSB signal. We could simply tune the local oscillator in Fig 8.30 to give us either the desired beat note for CW reception or to give us the frequency of the suppressed carrier in the case of an SSB signal. Whilst this would work, again there are benefits from using both the I and Q signals, as does the SDR receiver shown in Fig 8.27.

Let us analyse this further, by using some 'real' figures. If we assume that the wanted CW signal is at 14.101MHz and we would like to listen to a 1kHz beat note, we could tune the local oscillator to 14.100MHz since:

14.101MHz - 14.100MHz = 1kHz

However, if we have an additional, unwanted, CW signal at 14.099MHz, this also produces a 1kHz beat note since:

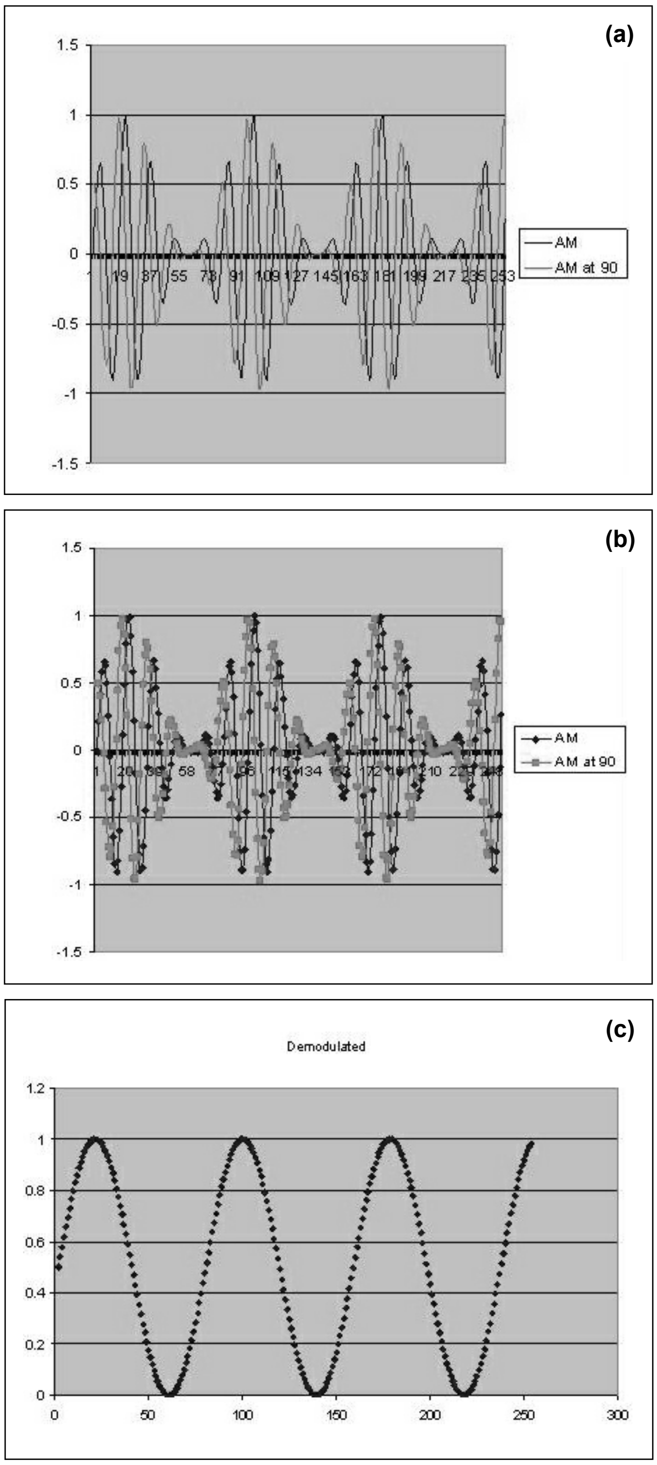


Fig 8.32: (a) AM I and Q signals at the input to the sound card; (b) Sampled I and Q AM signals; (c) AM demodulation using I and Q

14.100MHz - 14.099MHz = 1kHz

If we look at the I and Q waveforms that result from applying a 14.101MHz signal to the SDR receiver shown in Fig 8.27, we see two one-kilohertz sine waves with 90° phase difference - see **Fig 8.33(a)**. Note that the Q signal leads the I signal in time.

If we now feed a 14.099MHz signal into the receiver, we again see two 1kHz sine wave with 90° phase difference (**Fig 8.33(b)**). However, note that the Q signal now lags the I signal in time (the phase of the Q signal has been shifted by 180°). It is this phase

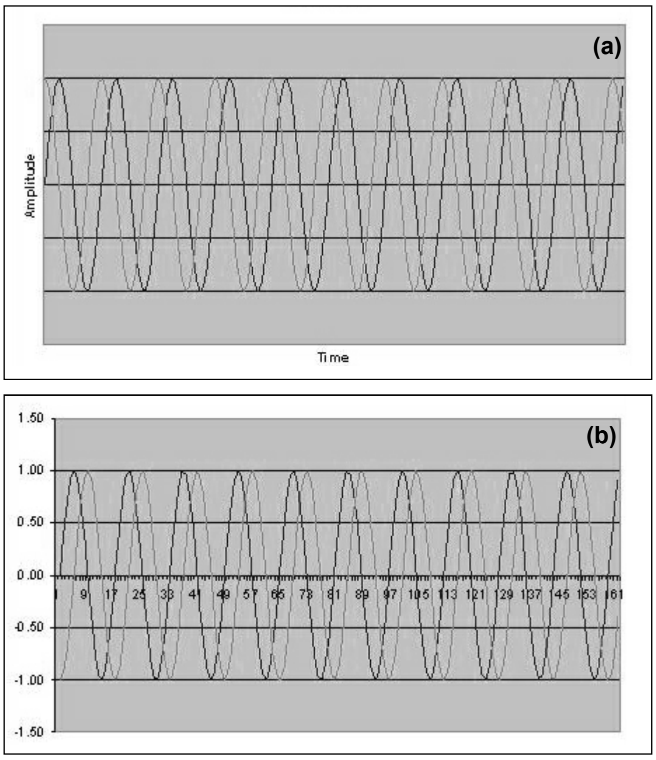


Fig 8.33: (a) Wanted signal I & Q phase relationship; (b) Unwanted signal I & Q phase relationship (Q is the lighter coloured waveform)

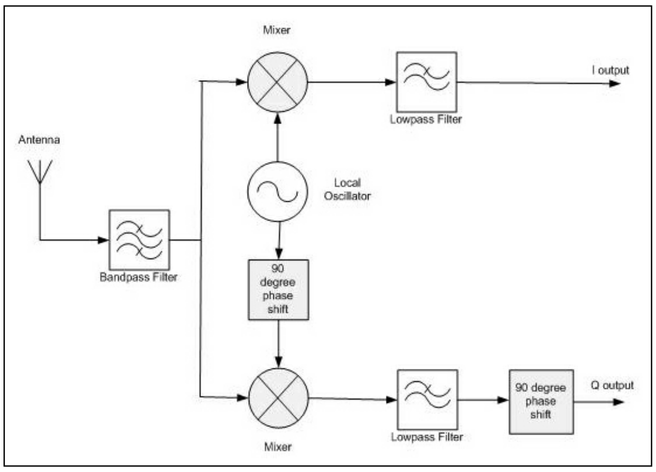


Fig 8.34: SDR front-end with Q phase shift

shift that enables us to remove the unwanted signal, as follows. In **Fig 8.34**, the I and Q signals have been passed to two low pass filters. Each has the same frequency response but the phase of all signals passing through the Q filter are shifted by 90°. **Fig 8.35(a)** shows the resulting I and Q signals for the wanted 14.101MHz signal, while **Fig 8.35(b)** shows the resulting I and Q signals for the unwanted 14.099MHz signal.

If the I and Q signals are now added together, the result is that we end up with a signal of double the amplitude of I or Q in the case of the wanted signal (**Fig 8.35(c)**), and zero amplitude in the case of the unwanted signal (**Fig 8.35(d)**).

The end result is that we have produced a receiver that responds to signals above the local oscillator frequency (ie Upper Sideband or USB) and rejects those below (ie lower sideband or LSB). If instead of adding the I and Q signals together we subtract them, then the reverse applies.

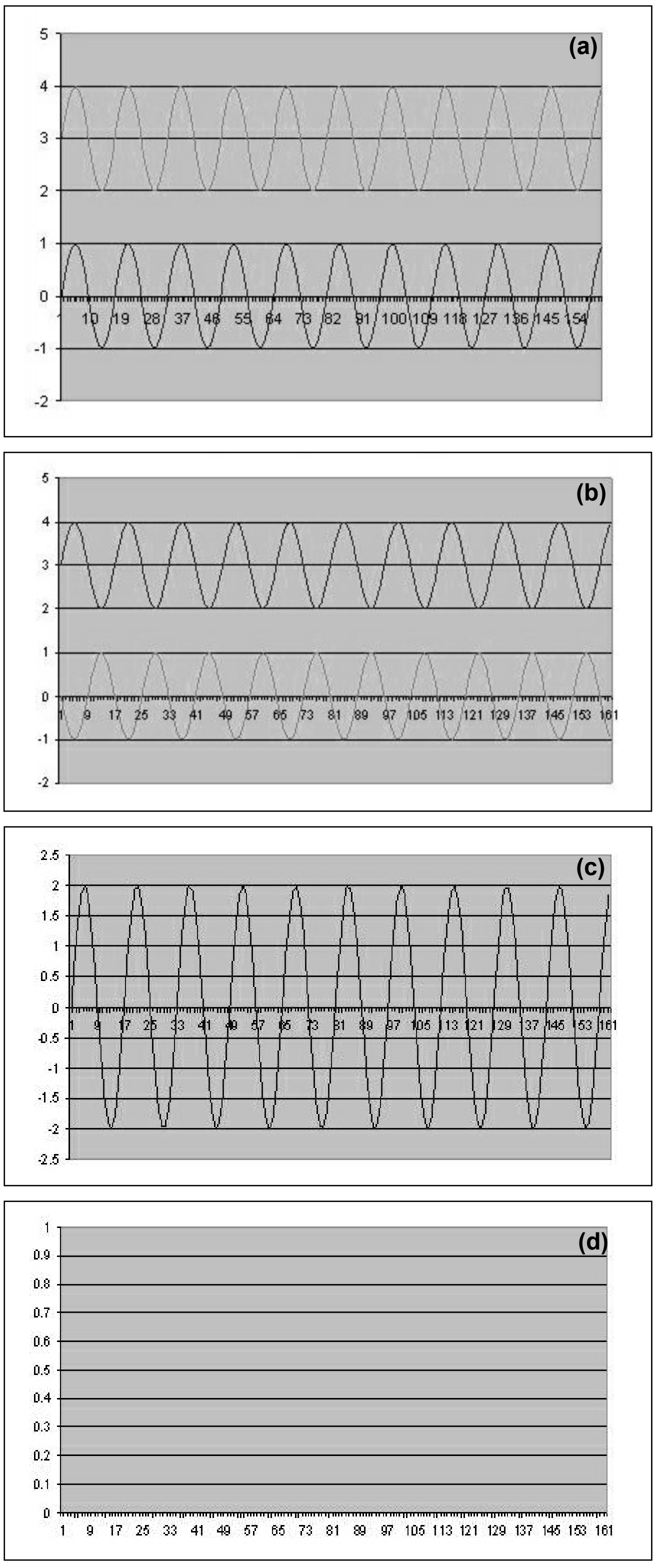
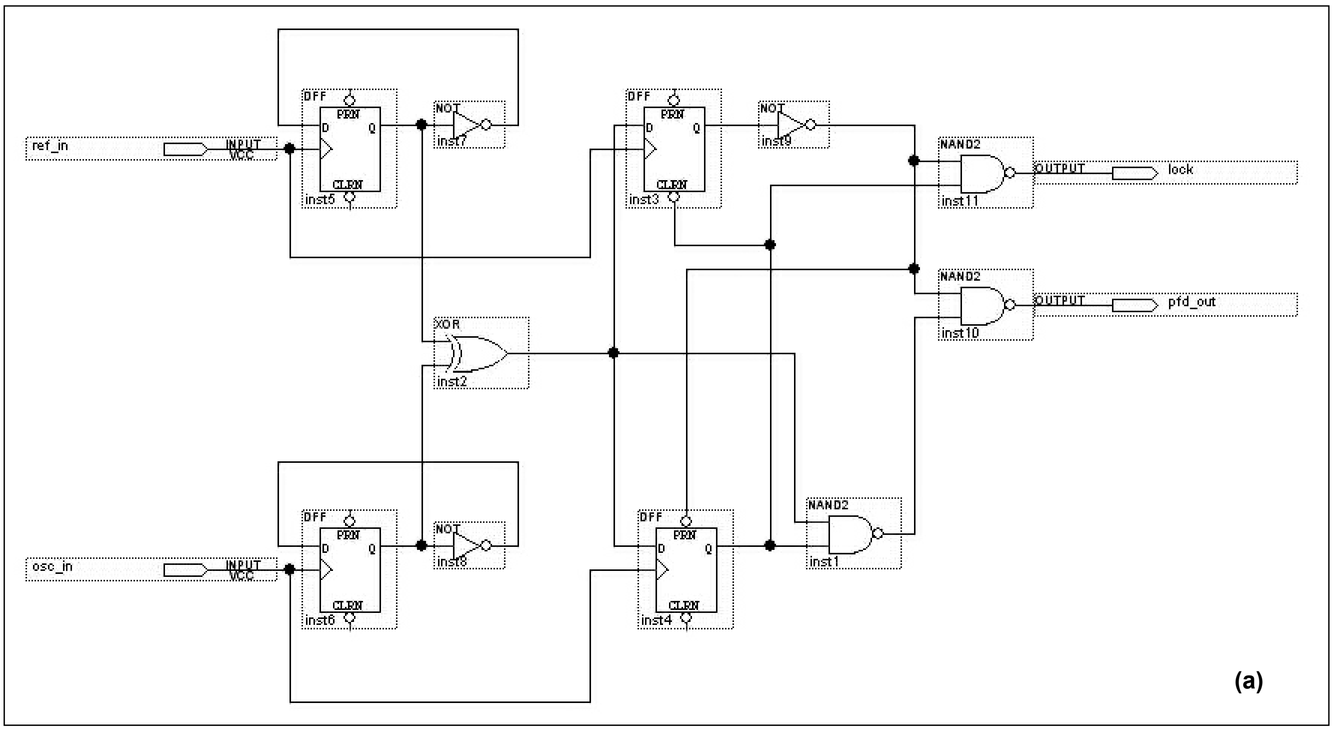


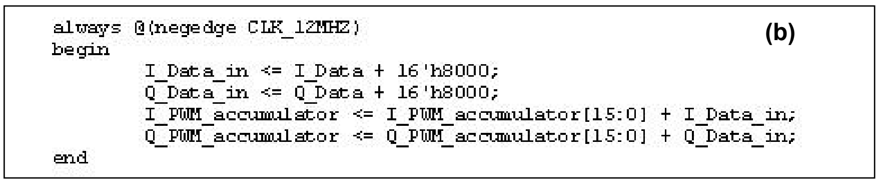
Fig 8.35: (a) I & Q phase relationship for wanted signal; (b) I & Q phase relationship for unwanted signal; (c) Result of adding I & Q for wanted signal; (d) Result of adding I & Q for unwanted signal

Phase and Frequency Modulation

To round off on demodulating the popular analogue modes using our SDR receiver, given I and Q signals we can demodulate a Phase Modulated signal using:

$$PM = \tan^{-1}(Q/I)$$





```
always @(negedge CLK_12MHZ)
begin
        I_Data_in <= I_Data + 16'h8000;
        Q_Data_in <= Q_Data + 16'h8000;
        I_PWM_accumulator <= I_PWM_accumulator[15:0] + I_Data_in;
        Q_PWM_accumulator <= Q_PWM_accumulator[15:0] + Q_Data_in;
end
```

Fig 8.36: (a) Phase-frequency detector by VK6APH as might be programmed into an FPGA (b) Fragment of Verilog code from an oversampling DAC

And for Frequency Modulation

$FM = (Q_n \times I_{n-1} - I_n \times Q_{n-1})/(I_n \times I_{n-1} + Q_n \times Q_{n-1})$

where n = current sample and n-1 = previous sample

We hope you now have a better understanding of why generating these two signals is so useful. As you can see, the processing of I and Q signals is fundamental to the operation of an SDR.

THE KEY ROLE OF THE FIELD PROGRAMMABLE GATE ARRAY

One technology that has been rapidly taken up by those experimenting with SDR is the Field Programmable Gate Array or 'FPGA'. It is convenient to think of an FPGA as a hardware integrated circuit whose internal function can be configured, and altered, by downloading software into the device. For those of us who cut their teeth using the TTL 7400 series of integrated circuits (ICs), this is an interesting development. Rather than keeping a stock of the various kinds of ICs - for example, AND gates, counters, etc - we can use an FPGA and determine its function by downloading the relevant software into the device.

We can also connect the inputs and outputs of the gates we have created within the FPGA using software. Rather than using physical wires and a soldering iron, this software configuration can be done by drawing a connection on the screen of a PC or by writing a few lines of code. Rather than having access to individual logic gates, an FPGA will typically hold a specific configuration of gates called a Logic Element. The number of Logic Elements can vary considerably - from a few hundred in a small FPGA to well over a million in a large device.

As well as Logic Elements, some FPGAs contain dedicated complimentary functions such as memory, Phase Locked Loop (PLL) synthesizers and frequency multipliers, as well as high-speed interfaces.

The Advantages of FPGAs

An FPGA costing a few tens of pounds can be programmed to replace hundreds of equivalent discrete ICs, yielding both size and cost advantages. Since the 'wiring' between the devices that make up an FPGA is performed in software, rather than tracks on a circuit board, the circuit can be corrected, improved or modified without needing to physically change the board.

For the experimenter, a PCB containing an FPGA can be reprogrammed numerous times as a project evolves, or the same PCB design can be used for different functions. Whilst many of these projects could be developed using a low-cost microprocessor, the FPGA has one major advantage: it can perform multiple tasks in parallel - a real boon for SDR work.

There are a number of ways to design the circuit you want an FPGA to perform. You can draw the schematic of the circuit on a personal computer (using the appropriate applications software), write software or use a combination of these methods.

For those first starting to experiment with FPGAs, it is tempting to draw the schematic of the required circuit, since all of the symbols will be familiar. **Fig 8.36(a)** shows the schematic of a phase/frequency detector that is to be implemented in an FPGA.

The advantages of schematic entry are soon diminished once a design uses more than a handful of logic functions. The schematic can grow to cover many pages on the personal computer screen, making understanding the FPGA's operation difficult - as well as complicating the inevitable debugging process.

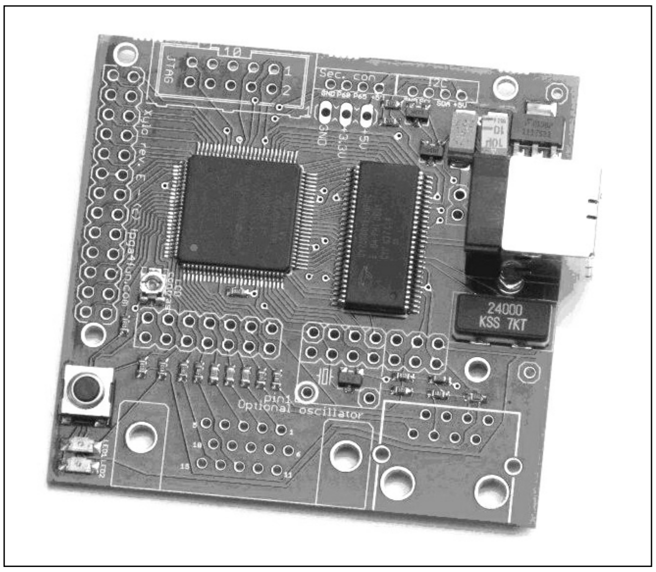


Fig 8.37: A Xylo Board

Whilst it is possible to develop very complex designs using schematic entry, the beginner is strongly advised to take some time to learn one of the FPGA programming languages. These are generically referred to as Hardware Description Languages or 'HDL'.

There are two languages that dominate the FPGA industry, namely Verilog and VHDL (the latter translates to 'Very high speed integrated circuit Hardware Description Language').

Verilog seems to be most popular in the USA, whilst Europe prefers VHDL. Having tried to learn both languages, VK6APH found Verilog much easier to learn since it has a lot of similarities with the C programming language. VK6VZ finds the very mention of either give him a bad headache.

Whilst the Verilog language is quite extensive, the FPGA implementation seems to use a relatively small subset. This enables the beginner to become proficient whilst only knowing a relatively small part of the language.

Fig 8.36(b) is an extract from the Verilog code used to implement a stereo over-sampling, Digital to Analogue Converter (DAC), used by the Janus card in the HPSDR project [8]. These seven lines of code are converted into some 40 logic gates in the FPGA. As you can see, this is a clear indication of the efficiency of writing in Verilog to configure an FPGA, rather than drawing the equivalent schematic.

In practice, the FPGA designer uses a combination of schematic entry and an HDL. For example, the phase/frequency detector in Fig 8.31(a) was copied from the data sheet of an existing IC. At the press of a key, this was automatically converted to Verilog code and included in the rest of the program.

The converse - being able to see how the HDL has been converted into logic gates - can also be useful when debugging a program.

There is also a considerable amount of free open-source Verilog and VHDL code available. This is fertile ground for the beginner, since 'example code' can be downloaded from the Internet, simulated and then loaded into an FPGA.

A good source of sample FPGA code can be found in the various lecture notes that the computing or engineering departments of numerous universities place on the Internet.

Kirk Weedman, KD7IRS, has recently developed a series of web-casts that present the basics of Verilog programming and much of the code used in the HPSDR project. You can view Kirk's presentations at [50].

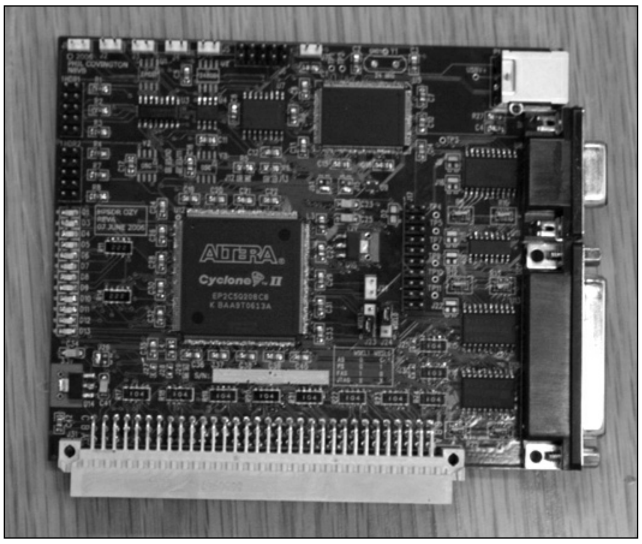


Fig 8.38: An Ozy board

Getting Started with FPGAs

So how does the beginner dip their toe in the pool of FPGA development? One way is to purchase one of the low-cost development boards that are produced specifically for this purpose. The two major FPGA manufacturers - Xilinx and Altera - both provide suitable boards, as do a number of smaller companies.

VK6APH's entry into the world of FPGAs came via a beginner's board called the Xylo [39] - see **Fig 8.37**. This board contains an Altera Cyclone EP1C3 FPGA, an FX2 microprocessor, full-speed USB interface (480Mbps) as well as Ethernet, VGA, I2C and RS232 capabilities plus some 38 additional input/output pins.

Since the Xylo board contains its own voltage regulator, no external supplies are required to use the board as it is powered from 5V via its USB connection to a personal computer.

The Xylo board comes with example C and Verilog code so as to enable the various communications and interface facilities to be fully investigated by the user. Whilst more suited for learning and simple projects, it should be noted a Xylo board was used to develop the entire Ozy_Janus FPGA code for the HPSDR project [8].

An alternative is the Ozy board [46] - see **Fig 8.38** - that forms the FPGA backbone to the HPSDR project. This uses a physically larger FPGA (an Altera Cyclone II EP2C5Q208C8) so as to provide more inputs and outputs, together with other features.

Since the HPSDR project is open source, both for software and hardware, all the circuits and driver code, etc, are freely available from the HPSDR website.

Either the Xylo or the HPSDR Ozy board will make an excellent introduction and learning platform for anyone interested in working with FPGAs. The HPSDR Ozy does have the advantage of being open source and intended for use with SDRs - this is marketed by the Tucson Amateur Packet Radio group (TAPR) [10].

Both Xilinx and Altera provide free development tools that can be downloaded from the web and used to develop code for a particular FPGA. In fact, testing the code on a physical FPGA is often the last thing that a developer does.

The development tools enable the code to be simulated running on the target FPGA and include the ability to attach virtual signal generators to input pins, and attach multi-trace oscilloscope probes to outputs, as well as to various test points within the program.

Fig 8.39 shows the stereo over-sampling, digital to analogue converter (DAC) (used by the Janus card in the HPSDR project) being simulated using Altera's *Quartus II™* software.

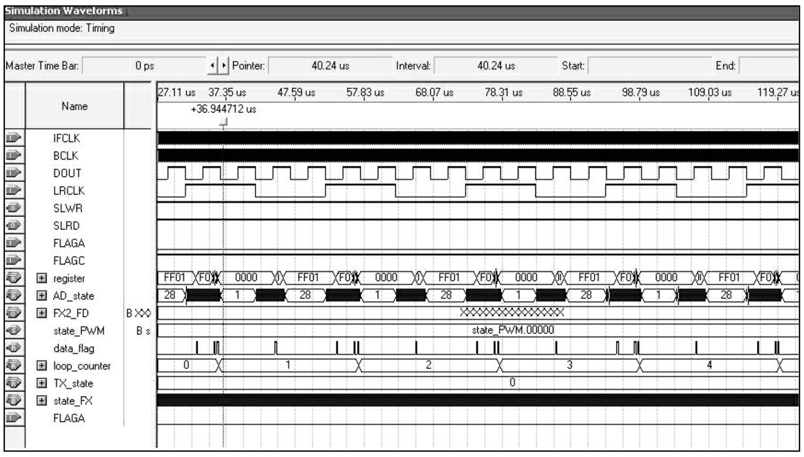


Fig 8.39: Screenshot of stereo oversampling DAC, simulated in Altera's *Quartus II*™ software

As the beginner will soon learn, just because a design works in the simulator does not necessarily mean that it will work when loaded into an actual FPGA! However, as one moves up the learning curve, this inconvenience happens less often.

When power is applied to a FPGA, it does not normally contain any code - the necessary code needs to be loaded into the device by some external means. For a completed project, this is often done from a small EEPROM (erasable programmable read-only memory) attached to the FPGA. For projects that are in development, there are many other options including JTAG, serial, parallel and USB methods of loading the code. There are also FPGAs that are based on Flash memory technology that retain their code when power is removed from the FPGA.

One of these is used in the Janus ADC (analogue-to-digital converter) and DAC board in the HPSDR project. This is a 'sister' device to an FPGA called a Complex Programmable Logic Device (CPLD).

CPLDs are generally small FPGAs that are used to 'glue' various other chips together or to replace a number of discrete ICs. VK6APH and VK6VZ regard the use of 'complex' as part of the name of CPLDs to be an unfortunate choice of terms and potentially discouraging the would-be user - whilst CPLDs are complex in the number of gates they contain, they are relatively simple devices to understand and use.

The particular CPLD used on the HPSDR Janus board is an Altera EPM240T that provides for 240 logic elements and uses Flash memory technology, so that it only requires initial programming. The reason for the use of a CPLD on the Janus board will be explained below.

REPLACING THE PC SOUNDCARD AS AN A- D CONVERTER - THE JANUS BOARD

One of the major factors in the performance of most of the first generation of software defined radios (what is often termed Quadrature Switching Detector or QSD architecture) is the a PC sound card that is used to digitize the analogue I and Q signals produced by the radio hardware/front-end.

In addition, a sound card with four input and four output channels is also necessary for a contemporary SDR transceiver (such as Flex-Radio's SDR-1000) in order to implement functions such as VOX (voice operated switching) and monitoring of transmitted audio or CW signals.

Many of the consumer and 'prosumer'-grade sound cards available in the personal computing market are quite good for these purposes. Of these, the M-Audio Delta 44 is probably the best known and most widely preferred, with its four inputs, four outputs and 24-bit sampling at rates up to 96kHz.

However, the sound cards available on the market today have some limitations when it comes to using them as analogue-to-digital converters and digital-to-analogue converters for SDR. For example, few of the prosumer and consumer cards on the market today can sample at a greater frequency than 96kHz. In addition, these cards often have filtering on their inputs and outputs that start to roll-off above 20kHz - definitely not desirable when carrying out sampling for an SDR!

Another problem with these PC sound cards is that only a handful that are capable of sampling above 96kHz use the popular USB connection method.

Given these limitations, in November 2005 a number of SDR enthusiasts including VK6APH and Bill KD5TFD decided to design and build a sound card specifically for SDR applications, as part of the High Performance Software Defined Radio (HPSDR) project [8].

The goals of the board, which was called 'Janus', would be to provide:

- 192kHz, 24-bit sampling;
- High signal-to-noise ratio - at least as good as the Delta 44;
- Single connection to the computer - eg using the high speed USB 2 interface;
- Full duplex;
- Four input and four output channels to provide VOX and transmit monitoring facilities; and
- Facilities to handle press-to-talk transmit and (possibly) RF hardware control.

In addition, building a dedicated sound card like Janus would mean that the designers would have complete control over any software drivers that were needed to communicate with the A/D chips, as well as being to potentially optimise sampling rates and bit depths for individual signals.

The basic design of Janus called for an ADC for I and Q signals and a DAC for transmitting I and Q signals, as well as an ADC for microphone input and a DAC for received audio. In order to connect to these various converters, 'glue logic' or a microcontroller was required.

Additionally, a USB interface was needed to send the digitized data to the PC and to receive the processed audio from the PC that needs to be sent to the DACs. A block diagram of Janus is shown in Fig 8.40.

To meet these design goals, a high quality ADC was necessary for digitizing the I and Q signals from the RF 'down-converter' hardware. After looking through the specifications of many high-end audio A/D chips, the design team decided to investigate the Wolfson WM8785, Texas Instruments PCM 4202 and the Cirrus Logic CS 5381. All of these devices are 24-bit, capable of sampling up to 192kHz and claim a 110dB, or better, dynamic range.

After breadboard testing these chips at a 192kHz rate, it was discovered there were some noise issues, apparently due to their design aliasing any noise above the audio frequency range in order to provide sampling at 192kHz. Whilst all of these devices will perform extremely well in an audio application, for SDR use the noise above ±48kHz would intermodulate with in-band signals and render a receiver unusable.

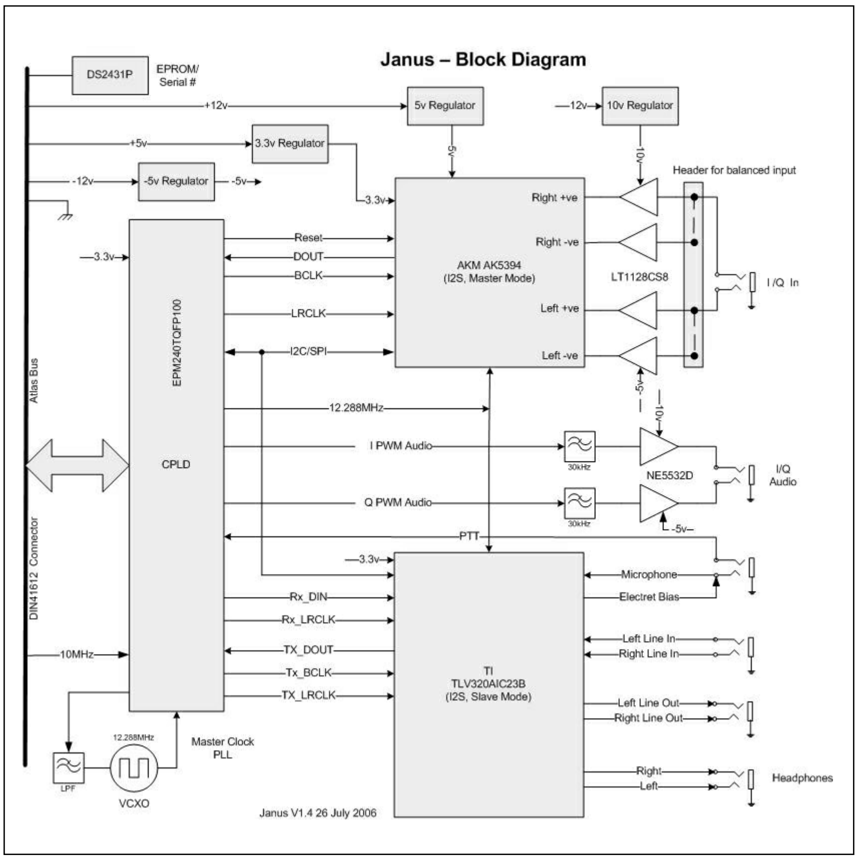


Fig 8.40: Block diagram of Janus board

As a result, after a suggestion from Bob McGwier, N4HY, an AKM AK5394A was then evaluated and chosen for the 24-bit A/D role in Janus.

The team also needed an ADC for a microphone input and a DAC and audio amplifier for the received audio output. Luckily, the dynamic range, bit depth and sampling requirements on these converters are not as stringent as that required on the I and Q converter, so the team chose the TLV320AIC23B - a highly integrated stereo audio CODEC-type 'chip' with a built-in headphone amplifier - for this purpose, operating at 48kHz sampling rate and 16-bits.

Intended for the MP3 player mass-market, this device provides totally adequate performance at a very low price (around US$7 at the time of writing).

For the DACs driving the I and Q outputs, a single bit over-sampling configuration was used. By sampling at 48kHz, a simple low pass filter is able to provide sufficient attenuation of the sampling frequency and minimize any phase difference between the two channels. The final design used a 16-bit over sampling single bit DAC, implemented in an Altera FPGA.

In order to interface the ADC and DACs to a graphical user interface (GUI) program running on a personal computer, a USB 2 interface was selected. Although USB 2 is specified to operate at 480Mbps, other experimenters using USB 2 for a similar function reported being able to obtain a maximum combined (ie simultaneous input and output) transfer rate of 240Mbps.

The fastest transfer rate occurs from the Janus board to the personal computer. At a 192kHz sampling rate, this consists of two-times 24-bit I/Q data, plus 16-bit microphone data, which requires a sustained transfer rate of approximately 12Mbps - well within the practical rates reported by others.

The designers chose to implement the USB interface using the Cypress FX2 family of devices, which provide a USB 2 interface and an 8051 microprocessor in the same chip. These devices have been used successfully in a number of similar projects and are well supported with open source code and development tools. As mentioned previously, the Janus board forms part of the HPSDR project. All the HPSDR boards plug into a standard backplane (called 'Atlas') that uses DIN41612 connectors.

As Janus was the first card to be designed following the design of the Atlas bus, it was considered unwise to define - and fix - a considerable number of bus signals at such an early stage in the project. In order to provide total flexibility for the Janus pin definitions, and to enable multiple Janus boards to be connected to the Atlas bus, an Altera Max II Complex Programmable Logic Device (CPLD) was used for the interface.

The CPLD basically provides a software configurable 'patch panel', allowing any Janus signal pin to be connected to any signal pin on the Atlas bus. At US$6 each, the CPLD was considered to be cheap insurance for pin assignments that will most certainly be altered in the future.

Design Considerations

Given the full duplex data requirements of Janus and the desirability of parallel processing, an FPGA solution was selected to interface to the FX2 USB chip. The FPGA and FX2 USB chips are contained on a second PCB called 'Ozy' that will be described in the next section of this chapter which deals with PC to SDR communications.

Prototype development of Janus was undertaken using an FPGA development board (manufactured by Jean Nicolle) called a Xylo board [39]. This consists of an Altera Cyclone EP1C3T100 FPGA interfaced to an FX2 chip and provides a USB 2 interface to a personal computer.

The *PowerSDR*^{TM} [12] software was modified to use the Xylo USB 2 interface for audio input and output. USB bulk transfer mode was used, in the hope that at USB 2 speeds (480 Mbps) isochronous mode would not be needed to achieve acceptable performance. This turned out to be correct and simplified initial development of the code necessary to 'talk' to the early Janus prototypes.

The Xylo board comes complete with drivers and sample code, which reduced the start-up learning curve and risks considerably. Since neither VK6APH or KD5TFD had any prior experience with FPGA development, nor programming in a high level definition language (HDL), the use of the development board seemed a good idea. It proved highly successful and very enjoyable.

VK6APH and KD5TFD chose to program the FPGA in the Verilog language rather than in schematic. Advice from those experienced in this art suggested that schematic design was easy for those with an electronics background but soon became unwieldy on larger designs.

At this point, Lyle Johnson, KK7P, undertook the design of a suitable PCB for Janus. One issue that had emerged during prototyping was that at 192kHz sampling, the 48kHz clock used for the TLV320 was visible in the Janus noise floor. However, with careful layout, and separation of analogue and digital ground planes, Lyle was able to eliminate the clock pick-up completely.

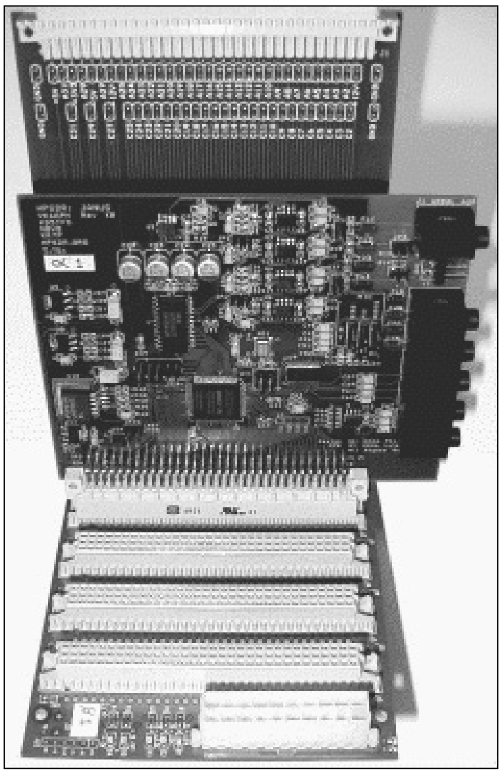


Fig 8.41: A Janus board plugged into the HPSDR Atlas bus and Pinocchio extender board

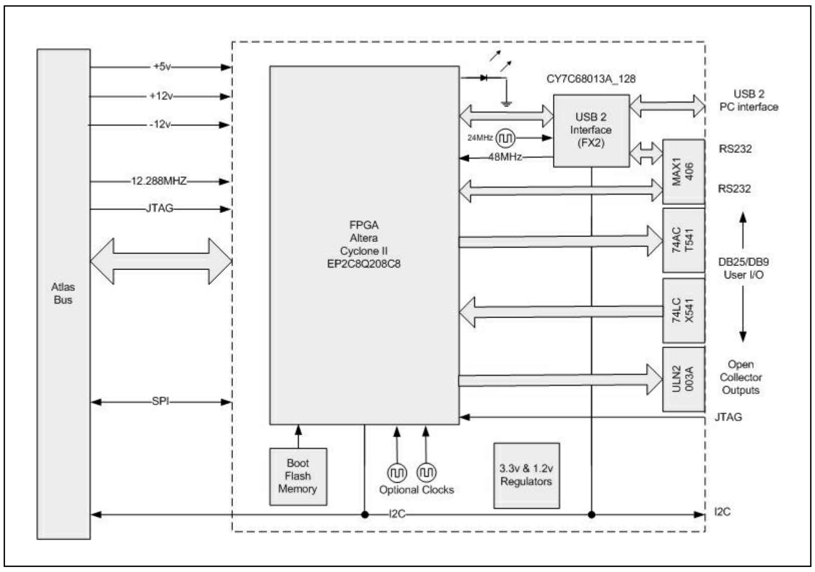


Fig 8.42: Block diagram of Ozy board

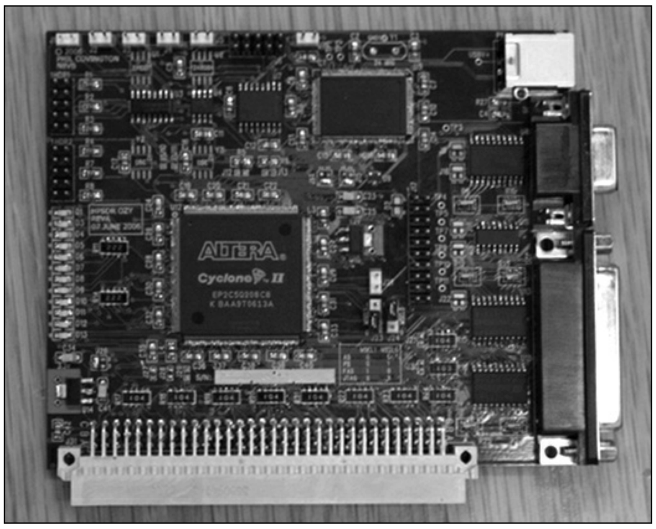


Fig 8.43: Prototype Ozy board

Whilst the Xylo FPGA was ideal for developing the code to interface the ADC and DACs to USB, it did not provide sufficient spare input/output pins for all the facilities required. Additionally, the Cyclone EP1C3T100 FPGA on the Xylo was somewhat limited in the number of RAM bits it provides. The RAM bits in the FPGA were primarily used as first-in-first-out (FIFO) memory to buffer the data going to, and from, the PC. VK6APH and KD5TFD found they were a little short on FIFO space at 192kHz sampling rates.

As part of the HPSDR project, as mentioned earlier, an FPGA based - USB 2 interface card, called Ozy has been developed. Originally christened 'Ozymandias', this has been designed and developed by Phil Covington N8VB, and provides the necessary FPGA resources, together with the requisite number of I/O pins required by the Atlas bus and sufficient RAM bits for buffer FIFOs.

Results

Measurements on Janus have been limited to comparing the signal-to-noise, noise floor and dynamic range against the M-Audio Delta 44 sound card. In all respects, Janus exceeds the already excellent performance of the Delta 44. A completed Janus card is shown in **Fig 8.41**.

One interesting discovery is whilst the AK5394A is specified as a 24-bit device, in reality it was only useable at 20-bits, the lower four bits apparently being simply noise. Whilst this may sound disappointing, this is consistent with professional sound cards in general - VK6APH points out that some of the 16-bit sound cards that are built into PC motherboards yield an Equivalent Number of Bits (ENOB) of only 12-bits!

As the result of the beta testing of the Janus PCB, a new feature was added, namely the ability to phase lock the 12.288MHz Janus master clock to a 10MHz reference. This in turn may be locked to a one pulse per second timing reference from a GPS receiver. The phase locking enables phase coherence of all oscillators in the HPSDR to be achieved. Production PCB files for Janus are available at [47].

PC TO SDR COMMUNICATIONS - THE HPSDR OZY BOARD

In the previous section of this chapter, we looked at replacing the PC soundcard as an analogue-to-digital converter (ADC) and how the HPSDR Janus board does this. Janus provides both very high performance ADCs and digital-to-analogue converters (DACs) for use with a SDR.

Although in the future the Janus board will be usable as part of a stand-alone SDR, on its own it is not able to communicate with a host PC. In order to send the digitised audio from the ADCs and DACs to, and from, the host PC in the HPSDR project [8] an additional interface board is required. For the HPSDR this board is called the Ozymandias (Ozy for short) - and its functions are shown in block diagram form in **Fig 8.42**. The prototype Ozy PCB shown in **Fig 8.44** was designed by N8VB.

There are several communications options for transferring the digitised audio data from the Janus board to the PC - these include Ethernet, USB and FireWire. Before deciding which of these options to consider, we need to work out the maximum data rate between the PC and the HPSDR hardware.

If we assume that for the time being that we will be using a fast sound card technology, then the currently available ADC chips have an upper limit of 192kbps, at 24-bits per sample. For the 'receive audio', microphone input and I & Q transmitter signals, a data rate of 48kbps, at 16-bits per sample, is quite sufficient.

Assuming we would like to operate the data transfer in full duplex, the mathematics is in **Table 8.2**. Having calculated these values, let's look at the speeds - and features - the various options provide.

Option 1: Ethernet

From practical experience, confirmed by a number of sources on the Internet, the actual speeds that the various Ethernet options provide is as follows:

10-T Ethernet:	specified at 10Mbps (1MBps in practice).
100-T Ethernet:	specified at 100Mbps (10MBps in practice).

Since most modern PCs support 100-T Ethernet, this looks like a possible option. The advantage of 100-T Ethernet is that it allows a long length of cable between the PC and an SDR (eg for mounting a microwave SDR on a tower). In fact, it is for this reason that the uWSDR group [40] has selected Ethernet as its interface standard.

Option 2: USB

USB supports a number of speeds as follows:

USB 1.1 Low Speed:	specified at 1.5Mbps (100kBps in practice).
USB 1.1 Full Speed:	specified at 12Mbps (1MBps in practice).
USB 2.0 High Speed:	specified at 480Mbps (30MBbps in practice and up to 40MBps with a very fast PC).

All modern PCs support High Speed USB 2.0 connections. The disadvantage of USB is that the cable lengths are restricted to between three and five metres, depending on the speed used, but for HF work where the SDR is located close to the PC this is not an issue.

The advantage of USB is that there is a plentiful supply of interface chips readily available, as well as open source drivers and sample code.

The fact that we can sustain 30MBps over USB will be an advantage in the future, as the HPSDR project adds high speed ADC boards to the range of options available. For this reason, USB 2.0 was chosen for the HPSDR project.

Option 3: Firewire

'FireWire' is officially known as IEEE 1394 and comes in the following speeds:

400Mbps:	in practice it runs faster then USB 2.0 since it uses less of the PC resources.
800Mbps:	no practical figures are available.

Many modern PCs have FireWire ports built into them and older PCs can be upgraded by adding a suitable PC card. The advantage of Firewire is the higher speeds available, but the disadvantage is that it requires a special built-in chip in the device, so it is more expensive to build into products. It has similar cable length restrictions to USB.

Other Requirements

Apart from the ADC and DAC data requirements we need to be able to send Command and Control (C&C) data between the PC and SDR hardware. This C&C data includes press-to-talk (PTT), bandpass and lowpass filter selection and SWR, etc. In comparison with the ADC and DAC data, the C&C requirements are of low volume and frequency.

The C&C requirements of the HPSDR project are met by sending data that requires low latency - eg PTT, dot and dash signals within the ADC and DAC frames and other signals via dedicated USB end points.

The other C&C functions provided by the Ozy board can be seen in Fig 8.42. These include an I2C master to provide communications to other I2C devices over the HPSDR's Atlas bus; two RS232 interfaces (one from the FX2 microprocessor and the other from the FPGA); and a parallel port (actually a printer port equivalent) that can be used to control SDR hardware (eg a Flex-Radio SDR-1000).

Atlas Bus Interface

Since the Ozy board needs to provide signal processing and PC communications for the Janus and other HPSDR boards, we need to consider how it will connect to the Atlas bus.

However, at what is a relatively early stage in the HPSDR project - and in full expectation that over time many other boards will be developed for the Atlas bus - it is difficult to 'set in stone' exactly what signals should appear on what line of the bus. Apart from the power supply voltages, which do need to be specified at an early stage, we need to be flexible in allocating signals to the bus.

In the previous section of this chapter it was mentioned that Janus used a complex programmable logic device (CPLD) to connect to the Atlas bus. The CPLD is used as a programmable 'patch panel', so that any signal required by, or sent out by, the Janus board could be patched to any Atlas bus pin.

This use of the CPLD provides great versatility in the future and would also enable multiple Janus boards to operate simultaneously on the one Atlas bus. This can be achieved by patching common signals to the same bus pin, eg clock signals and unique signals to separate bus pins. The use of an FPGA or CPLD to connect a board to the Atlas bus will be a common feature, as the HPSDR develops.

From Janus to PC						
2 x	192,000	x 24	=	9,216,000bps	(ie received I and Q signals at 24-bits)
4 x	48,000	x 16	=	3,072,000bps	(ie mono microphone at 16-bits(*note 1*))
		Total	=	12,288,000bps	(or approximately 1.5MBps)
From PC to Janus						
2 x	48,000	x 16	=	1,563,000bps	(ie stereo received audio at 16-bits)
2 x	48,000	x 16	=	1,563,000bps	(ie transmit I & Q signals at 16-bits)
		Total	=	3,072,000bps	(or approximately 0.4MBps)

NOTES:bps = bits per second, MBps = Megabytes per second).
Note 1: The data from Janus to the PC is sent in a packet that contains the received I & Q signals plus the microphone data. Since it is desirable that the packet length is constant, the microphone data is sent at four times its actual sampled rate of 48kHz and the sampling rate is corrected in the PC software.

Table 8.2: Full duplex data transfer for a fast sound card

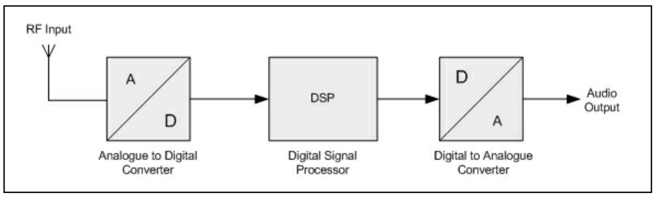


Fig 8.44: Ideal direct down conversion receiver

DIGITAL DOWN CONVERSION PRINCIPLES

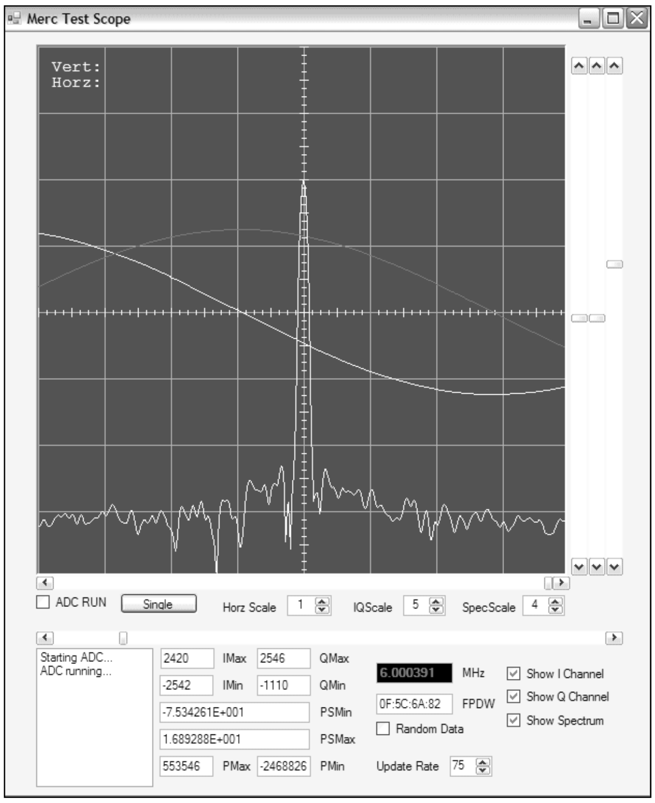
The 'holy grail' of Software Defined Radio is to be able to directly digitize a wide RF spectrum by connecting an antenna directly to an analog-to-digital converter (ADC). The signals are then digitally processed and converted back to analog with a digital-to-analog converter (DAC) for audio amplification purposes (see **Fig 8.44**).

At the moment, at least for other than very expensive and specialized military projects, this is not economically feasible. However, just like the speed of PCs has increased enormously over the years, in the future we can expect to see amateur receivers using this architecture.

To date, the majority of amateur SDRs designed for high frequency radio purposes have used PC sound cards as their ADC. In fact, by using semi-professional sound cards in a PC, an SDR with high performance can be realized. However, for those who write the history of SDR, this usage is likely to be a relatively short trend in the transition to direct digital conversion.

At present, the technical stumbling block to direct digital conversion is that even with the processor speeds of modern personal computer CPUs, there is still a processing bottleneck. However, by compromising our design so that we process a narrower band of frequencies rather than the entire HF radio spectrum, it is now actually possible to produce a cost effective, very high performance 'almost direct-digital-conversion' receiver, using what are called Digital Down Conversion (DDC) techniques. Earlier in this chapter, we covered some of the commercially available DDC receivers, plus a basic description of the HPSDR Mercury receiver, which uses this technique.

A key hardware item in such a DDC SDR is the ADC used for the RF front-end. Until recently, ADCs had either had too low a dynamic range to be useful on the HF bands or were too costly or both.



ADC RUN ☐ | Single | Horz Scale 1 | IQScale 5 | SpecScale 4

Starting ADC...
ADC running...

2420 IMax 2546 QMax
-2542 IMin -1110 QMin
-7.534261E+001 PSMin
1.689288E+001 PSMax
553546 PMax -2468826 PMin

6.000391 MHz
0F:5C:6A:82 FPDW
☐ Random Data
Update Rate 75

☑ Show I Channel
☑ Show Q Channel
☑ Show Spectrum

Fig 8.46: Screen shot of LT2208 used as a spectrum analyser by Phil Covington, N8VB

Luckily for amateur SDR enthusiasts, the massive take-up of cellular phone technology has produced a mass market for high performance ADCs that amateur constructors can benefit from.

One of the highest performance ADCs on the market at present is the Linear Technologies LT2208. At a one-off cost of U$100 it is not inexpensive, but its high performance in a DDC makes this price tag relatively easy to justify. After all, the ADC will form the major part of the DDC radio.

A block diagram of the LT2208 taken from the Linear Technologies data sheet is shown in **Fig 8.45**. The LT2208 provides 16-bits of data at a sampling rate of up to 130MHz. This provides enough dynamic range and speed to sample the entire radio spectrum from 0 - 65MHz in real time.

When connected to a PC via a high speed USB 2 connection, the LT2208 makes an impressive spectrum analyzer.

Phil Covington, N8VB, has already experimented with using the LT2208 as a broadband spectrum analyzer and a screen shot from his PC software is shown in **Fig 8.46**.

If we are unable to directly digitize the entire HF spectrum and process it in real time, what compromises do we need to make in order to make use of DDC technologies?

The answer is that basically we use superhet techniques where a band of frequencies are mixed down to a lower and narrow range of frequencies

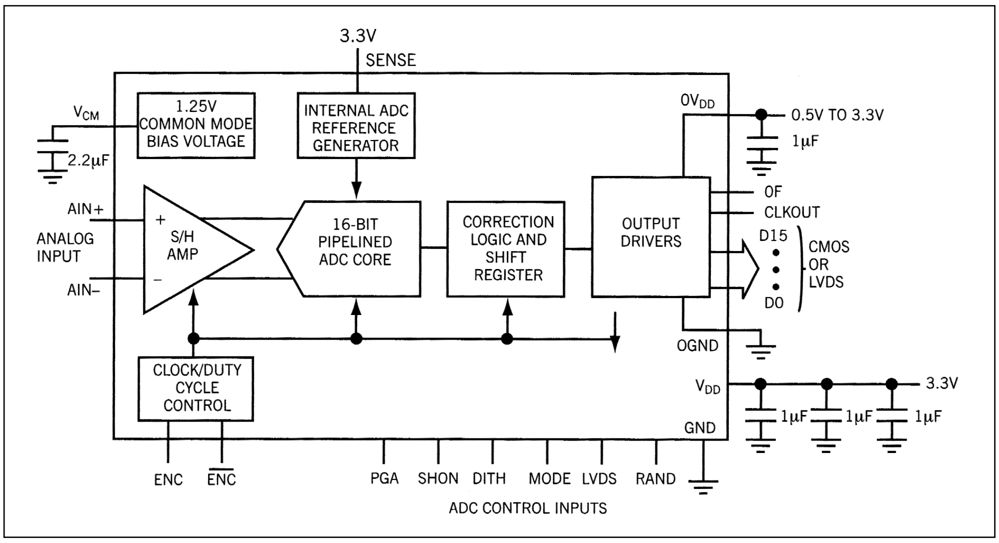


Fig 8.45: Functional block diagram of LT2208, reproduced from Linear Technologies Inc data sheet

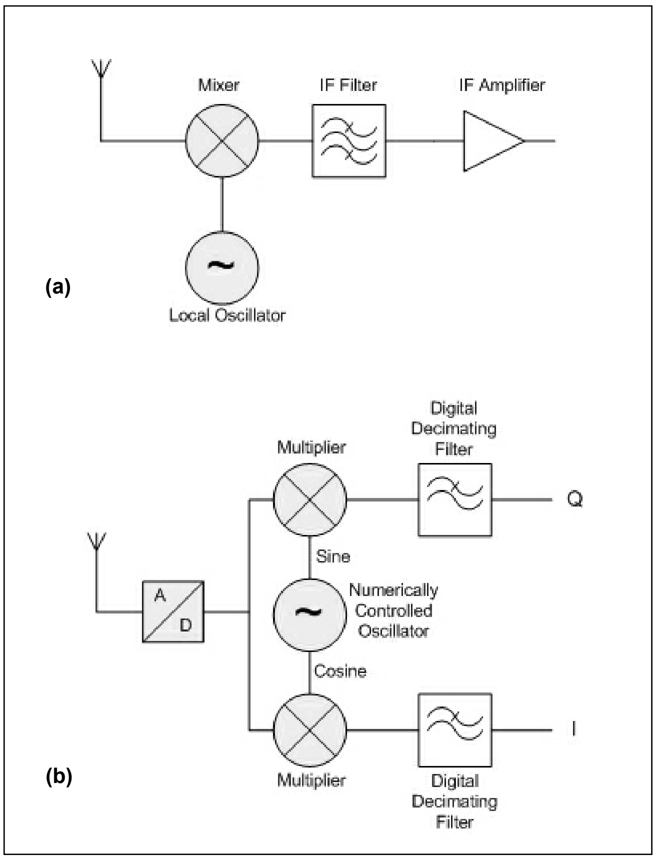

Mixer
IF Filter
IF Amplifier
(a)
Local Oscillator
Digital Decimating Filter
Multiplier
Q
Sine
A
D
Numerically Controlled Oscillator
Cosine
I
(b)
Multiplier
Digital Decimating Filter

Fig 8.47: Block diagrams showing a comparison between (a) an analogue superhet receiver and (b) a digital down conversion receiver

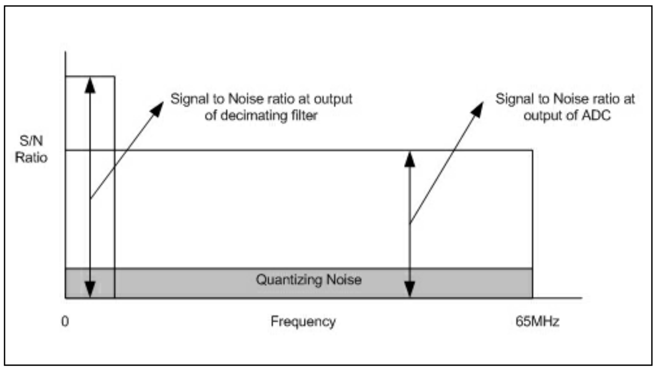


Fig 8.48: Decimation signal-to-noise (S/N) improvement

that are within the processing speed of the current generation of digital processors.

In order to make this idea perfectly clear, the analogy of the Intermediate Frequency (IF) stages in an analogue radio is very helpful. Compare the block diagram of an analogue superhet receiver to that of a DDC one - see Fig 8.47.

We will explain in detail how the digital local oscillator in the DDC-based receiver is generated in a later section of this chapter.

Additional information on this technique is available in an article by Analog Devices, 'Single Chip Digital Synthesis versus the Analog PLL' [48] and is highly recommended.

In the case of the analogue superhet, a local oscillator signal is mixed with the incoming signal, so that the sum or difference of these signals is equal to the desired IF. In contrast, in a DDC-based receiver a software oscillator is used to 'translate' the frequency band of interest down to a lower frequency band that can be sampled at a lower rate.

Since we wish to generate I and Q signals, two local oscillator signals at 90° to each other are used (ie a sine and a cosine signal).

In the analogue superheterodyne receiver design, the mixer is followed with a narrow band filter - either inductor and capacitor-type (LC), crystal, ceramic or mechanical. The purpose of the filter is to reject high level signals on either side of the wanted signal, so as to prevent these high level signals from overloading the following stages of the receiver.

Often the analogue 'superhet' uses a relatively wide 'roofing' filter at this stage, with narrower filters being used further down the IF strip, to keep out the high level unwanted signals. In the DDC receiver, we use a similar technique, although in this case the filtering is done digitally and the data speed reduced using a technique called decimation.

For example, if the DAC is sampling the signals at the antenna socket at, say, 130MHz and we need to reduce this data rate to, say, 130kHz, for subsequent transfer over a USB 2 connection to a personal computer, then the filter will decimate by:

130,000,000/130,000 = 1,000

The act of decimation is simple in the extreme; we simply throw away all but every 1,000th sample. There is also a most useful side effect to using this decimation technique - which is to increase the signal-to-noise ratio of the overall DDC by the decimation ratio.

Since the ADC converts the instantaneous RF input signal level to one of 2^{16} levels, there will frequently not be an exact level that matches the incoming sample. This error is referred to as a quantizing error and is responsible for noise at the output of the ADC.

The improvement in dynamic range can be explained by referring to Fig 8.48. Here the noise generated by the ADC is seen to be spread over the entire frequency range of clock/2, so with a clock of 130MHz this will extend from 0 - 65MHz.

If we only select a narrower band of these frequencies at the output of the DAC, then the amount of noise we decode is proportionally less.

Considering the above example, decimating by 1,000 will increase the dynamic range by:

10 log 1000 = 30dB.

This is the best theoretical improvement - although not all of this improvement in dynamic range will be realized in practice, it is still a most useful feature, as we will see.

If the ADC in our DDC is an Linear Technologies LT2208, then based on the specification of this device the overall performance of an HF receiver using DDC with this as its ADC can be calculated. If it is assumed that the input impedance of the LT2208 is 50 ohms and the maximum input signal level (derived from the LT2208 datasheet) is 2.25V peak-peak, then this translates to +11dBm.

The IP3 (level of third-order Intermodulation Products generated) of the LT2208 device is specified at -1dB below maximum input (ie +10dBm).

From the datasheet, at this power level the maximum spurious level is -85dBm. If the graphs provided in the datasheet are used, this translates into an IP3 of +54dBm - a very respectable figure indeed for a radio receiver.

The signal-to-noise ratio of the ADC converter is typically 75.2dB at 30MHz and 75.3dB at 5MHz, suggesting that the noise can be taken as being evenly spread across the sampling bandwidth.

Normally with a single ADC, the Nyquist bandwidth is half the sampling frequency but we will be generating in-phase and quadrature (I and Q) signals so the noise is spread across the full sampling bandwidth (ie it can be thought of as two samples).

The noise then will be bandwidth-limited in the signal processing, thus reducing the noise referred back to the input of the ADC by the ratio of the sampling bandwidth to the final bandwidth (ie the decimation gain).

If the LT2208 clock is run at 100MHz, then for a 500Hz CW bandwidth the noise referred ADC is equal to:

= -75.2dB below -1 dB FS - 10 Log(100 MHz/500Hz)
= -128.2dBm

This equates to a noise figure of 19dB. However, if we ensure that the external received noise is 10dB above the (internally generated) receiver noise, the internal noise will only add 0.46dB to the received noise floor.

In practice, the external noise picked up by even the best radio receiver over the 0 -30MHz radio spectrum is going to be generally at least 10dB above its internal noise - and on amateur bands such as 160m or 80m, it is going to be well over double this level.

The maximum noise figure for a given frequency to meet this requirement has been determined by a number of authors [49]. Using these values, the following noise figure requirements for an HF Bands receiver can be calculated- shown in **Table 8.3**.

Band	External noise (dB above kTB*)	Noise figure (dB)
80m	38	28
40m	33	23
20m	28	18
15m	23	13
10m	18	8

NOTE: External noise is measured professionally in receivers in “dB above kTB” (where k is Boltzman’s constant, T is 288K and B is bandwidth).

Table 8.3: Noise figure requirements for an HF (amateur bands) receiver

As can be seen, the noise figure calculated for the LT2208-based DDC receiver is acceptable in practice for all amateur bands below 15m. In order to meet the noise figure requirements on 10m, it is necessary to add a conventional pre-amplifier to the receiver.

If it is assumed a pre-amplifier can be built with a 3dB noise figure, then this will need to have a gain of 12.5dB to give an overall noise figure of 8dB.

If it is assumed the pre-amplifier has an input IP3 of +35dBm - an IP3 which most good current designs can provide - then this would match the system well and yield an overall IP3 of +33dBm.

In summary, the overall performance of the LT2208-based DDC receiver is shown in **Table 8.4:** The good news is that the HPSDR [8] Mercury receiver board which uses the LT2208 ADC has shown that these figures can be achieved in practice.

	Without pre-amplifier	With pre-amplifier
Noise figure	19dB	8dB
Input IP3	54dBm	33dBm
Maximum signal	s9+80dB	s9+68dB

Table 8.4: Performance of the LT2208-based DDC receiver

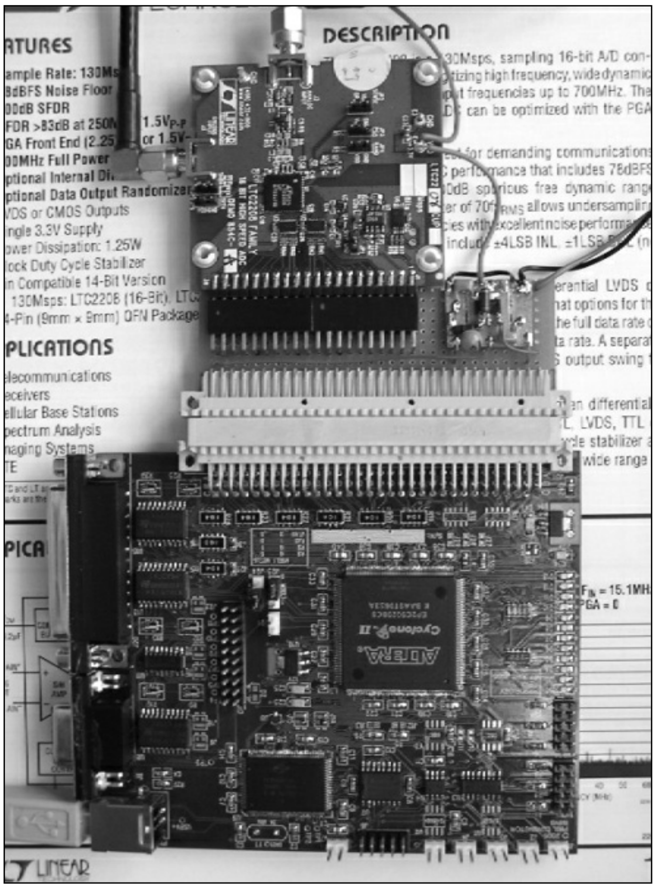


Fig 8.50: Prototype of the Mercury receiver (with HPDSR Ozy board attached)

THE HPSDR 'MERCURY' DIGITAL DOWN CONVERSION RECEIVER

Basic Operation of Mercury

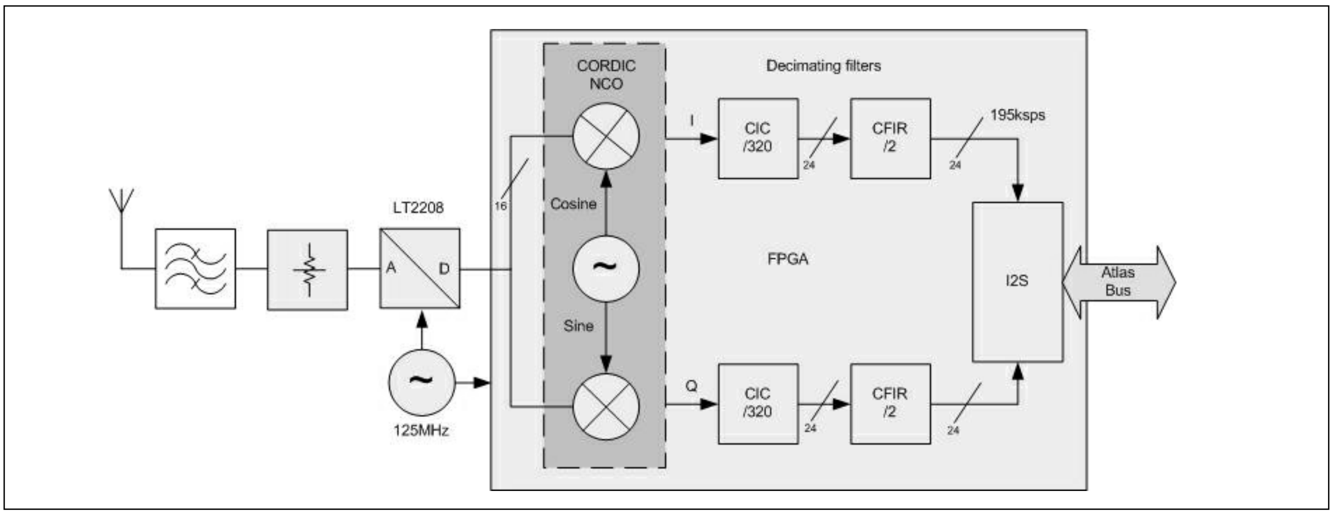
The 'Mercury' receiver design is able to receive signals over the range 10kHz to 55MHz and uses the popular *PowerSDR*^{TM} [12] PC software for demodulation and control. A photo of the initial Mercury prototype by VK6APH is shown in **Fig 8.49**. In this section, we will go into detail regarding the design of Mercury since we firmly believe that the Digital Down Conversion (DDC) approach used in it is the future of amateur radio receivers.

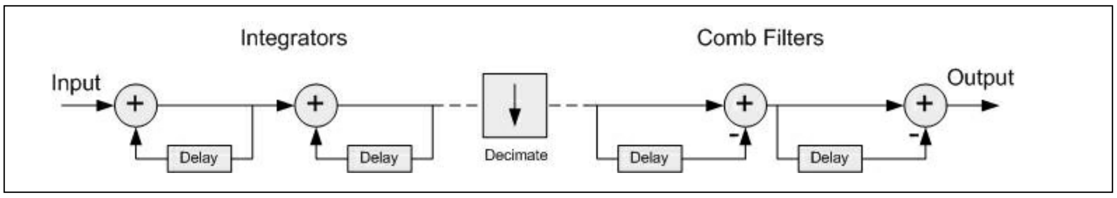
A block diagram of Mercury is shown in **Fig 8.50**. A signal received at the antenna is first 'band limited' and then optionally attenuated or amplified, depending on the frequency band in use.

This signal is then fed into an LT2208 ADC that produces a 16-bit digital output stream at 125Msps*. This data stream is in turn fed to a 'CORDIC' Numerically Controlled Oscillator or NCO. The operation of a CORDIC NCO is described later, but for now readers should consider it as equivalent to the local oscillator and mixers in a direct conversion receiver.

The output from the CORDIC NCO is in the form of two data streams - the I & Q streams that have been mixed down to 'base band' (ie to a range of frequencies equal to the RF input minus

NOTE: * *Msps = Millions of samples per second*
** *ksps = Thousands of samples per second*





(above) Fig 8.50: Mercury receiver block diagram

Fig 8.51: Block diagram of a decimating CIC filter

the local oscillator frequency). In the case of Mercury, a passband of 96kHz is used, which will result in a bandscope width of 192kHz.

At this stage, the data comprising the I and Q signals is still 16-bits wide at 125Msps - too fast to be processed economically by a signal processing chip or a personal computer. This data rate needs to be reduced to a more manageable speed - this is done using a process called 'decimation'.

For example, in Mercury we initially decimate by 320, which drops the data rate from 125Msps** to 125/320 = 390ksps**. To decimate, the unwanted samples are simply discarded, hence we simple keep each 320th 16-bit sample and ignore the rest. Decimation also has the effect of increasing the signal-to noise ratio and to use this additional range, the number of bits per sample grows from 16 to 24.

The resulting data stream now needs to be filtered in a similar manner to that of the roofing filter of an analogue receiver. Whilst there are numerous digital filtering techniques that could be used here, most SDR designers use a filter called a 'Cascaded Integrator Comb' or CIC.

The big advantage of the CIC filter is that it can be implemented without the use of multipliers - these tend to be at a premium in silicon-based signal processors. Instead, the CIC uses addition and subtract functions, which are usually plentiful.

Another good thing about a CIC filter is that it also allows us to combine the steps of filtering and decimation into one stage.

A block diagram of a CIC filter is shown in Fig 8.51. You can see why the filter gets its name if you look at the block diagram - the filter literally consists of cascaded integrators and comb filters.

The frequency response of a CIC filter is shown in Fig 8.52. In fact, this plotted response is of the actual CIC filter used in Mercury. The response consists of a series of nulls at integer multiples of clock/decimation. In the case of Mercury, these will be at a spacing of 125MHz/320 = 390kHz.

The overall shape of the CIC filter response is known as a sin(x)/x or sinc shape. As can be seen from the frequency response, the CIC filter also has a massive gain - some 400dB - as the result of using cascading integrators. This gain can be simple removed by truncating the output data width.

Whilst elegant in its simplicity - and frugal in its use of silicon - there are several drawbacks to the use of a CIC filter. The first is evident from the frequency response - ideally we would like a flat-topped 'brick wall' filter. As Fig 8.53 shows, the filter has about 6 dB of droop in its passband in the region we are interested in, namely 0 to 96kHz.

Secondly, the CIC filter suffers from significant 'alias' responses. Any signal within ± 96kHz of a null will be reflected back into the passband and appear as an unwanted spurious signal. It is important to understand that once an unwanted signal has been aliased back into the passband, then no amount of filtering past this stage can remove it.

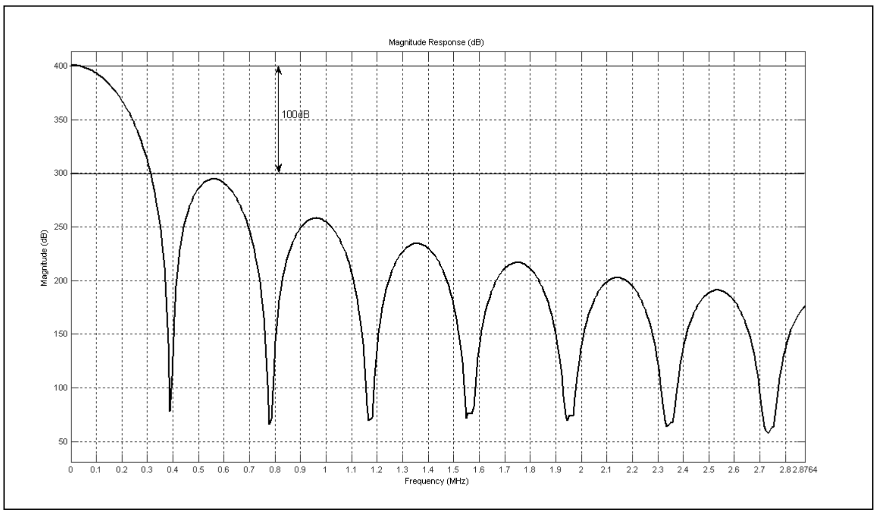


Fig 8.52: Frequency response of the Mercury CIC filter

The bands of frequencies where these alias signals can occur are shown shaded in grey in **Fig 8.54** - as you can see there are lots of them! We need to ensure that the level of alias signals that can be reflected into the passband are reduced to an acceptable level.

For Mercury, the spurious response level has been set by the designers to be 100dB below maximum input level. This means that at the worst case of (n x 390kHz) ± 96kHz, the filter must have an attenuation of 100dB.

For Mercury the only area of concern is around the first null at 390kHz, since the signals that will be aliased around all higher nulls are well below this 100dB level.

As can be seen from **Fig 8.55**, the filter designed for Mercury meets this requirement. The level of attenuation is achieved by using a filter of the appropriate 'order' - the order simply being the number of integrators and comb filters used. In the case of Mercury, this is an 8th order filter (ie it uses eight cascaded integrators, a decimator (divide by 320), followed by eight cascaded comb filters).

Higher alias attenuation can be achieved by using a higher order filter, but then the droop in the passband increases, so a compromise needs to be made between alias attenuation and passband droop.

Referring again to Fig 8.50, the CIC filter is followed by a more conventional filter; typically a Finite Impulse Response (FIR) design. Since the bulk of the filtering has been taken care of by the CIC, we can spare some valuable multipliers in the silicon to implement this filter. Rather than going off-topic here and explaining FIR filters, think of them as analogous to the main crystal filter in a conventional analogue radio.

The response of the basic FIR filter used in Mercury is shown in Fig 8.60 - as can be seen, it is most impressive. The -3dB point is about 92kHz and it reaches -110dB at 96kHz - a specification that most analogue filter designers would find extremely challenging! In fact, a slightly different FIR is used in Mercury and this is called a Compensating Finite Impulse Response (CFIR) filter. Rather than being flat, the passband of the CFIR has the opposite shape to the droop of the CIC that precedes it. When the two filters are cascaded, the droop is cancelled and a nice flat passband results.

The CFIR filter also decimates by a factor of two, bringing the final data rate - which is passed via USB2 to the personal computer - to a very manageable 195ksps at 24-bits per sample.

The overall frequency response of the cascaded CIC and CFIR filters used in Mercury are shown in **Fig 8.56**. This response has been achieved in the prototype Mercury receivers.

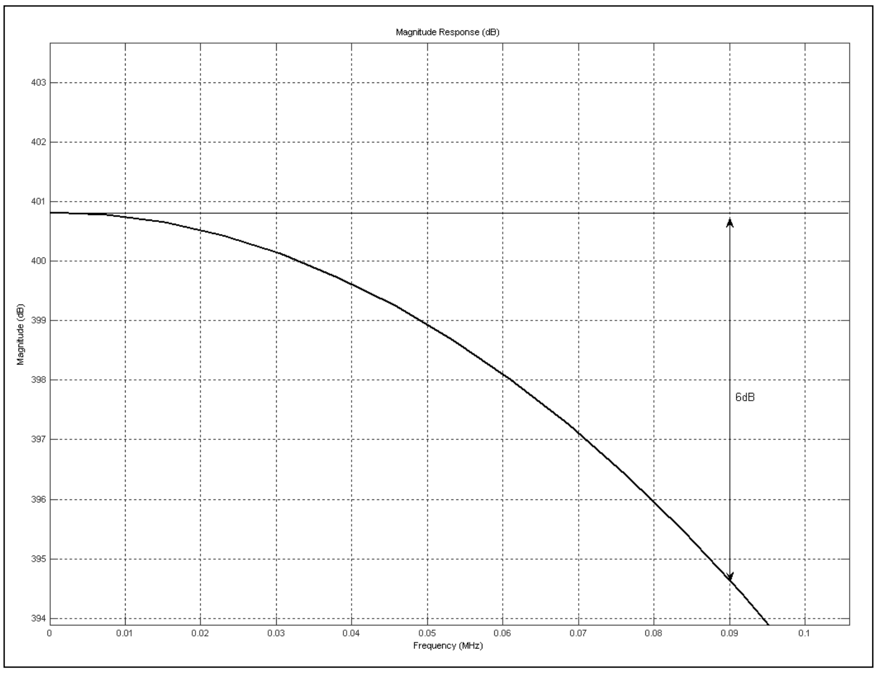


Fig 8.53: CIC filter passband droop

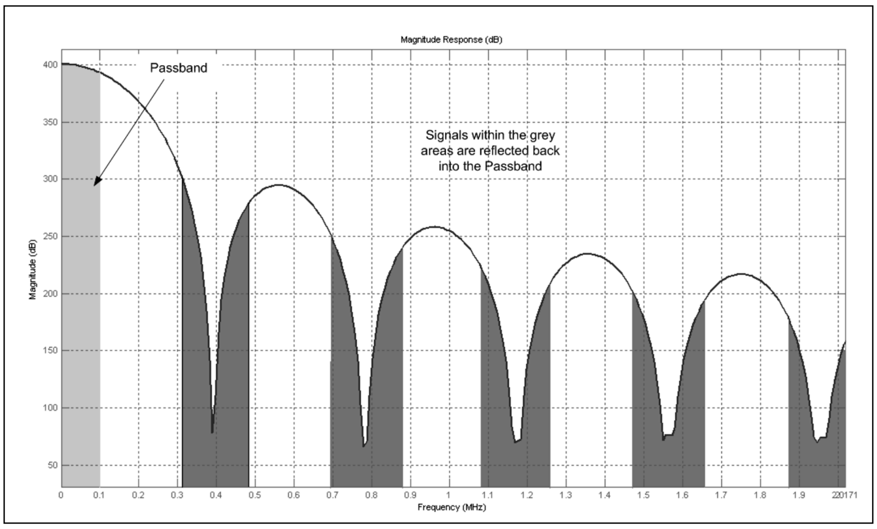


Fig 8.54: Alias responses of the Mercury CFIR filter

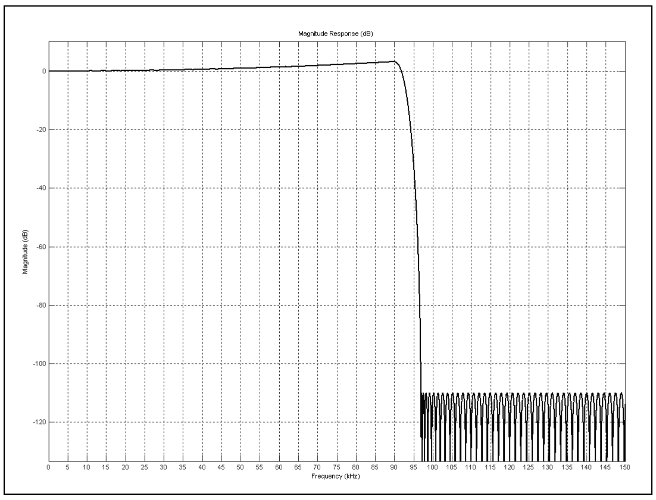


Fig 8.55: Frequency response of the Mercury CFIR filter

Mercury & Digital Direct Down Conversion

As explained in the previous section, in a Digital Down Conversion (DDC) SDR receiver such as the HPSDR Mercury, we need to mix the incoming RF signal down to a lower frequency in order for it to processed by a Digital Signal Processor (DSP).

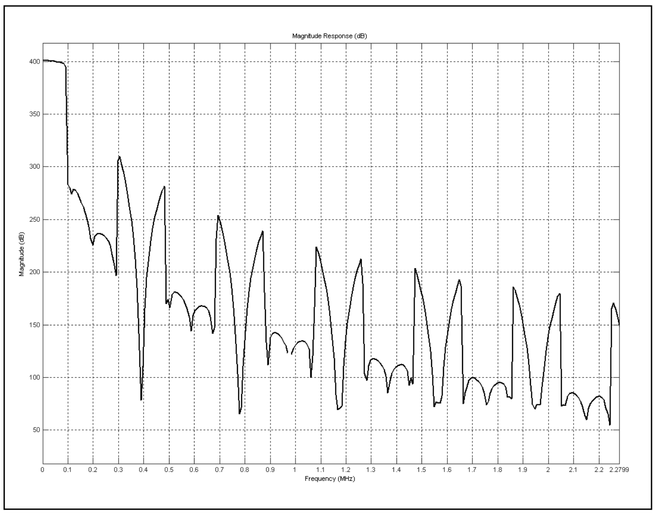


Fig 8.56: Overall frequency response of the Mercury receiver

The mixing process is similar to that used in a conventional analogue receiver to convert to a fixed Intermediate Frequency (IF). In the case of an SDR we need to produce two data streams - I and Q - in order to simplify subsequent demodulation.

Whilst conventional analogue techniques could be used (ie a local oscillator feeding two mixers) and then pass these mixer outputs to analogue-to-digital converters (ADCs), there are fully digital techniques that have significant advantages over the former.

One major advantage is that the generation of I and Q signals is for all practical purposes perfect. This means there is no need to provide phase and amplitude balance controls to null-out unwanted images on receive (or unwanted sidebands on transmit).

Let us assume that an RF input signal has already been passed to an ADC and it is desired to generate our I and Q signals digitally. This is achieved by multiplying the digital data from the ADC by sine and cosine at the local oscillator frequency (see **Fig 8.57**).

So how do we generate this local oscillator without the use of conventional analogue components (eg. inductors, capacitors, etc)? The technique most often used is Direct Digital Synthesis (DDS - see also the chapter on Oscillators) which works as follows.

In order to generate a sine wave, the value of the sine of a series of uniformly incremented angles is stored. For example, if we assume that our angle is incremented in one degree steps, then we can form the sine (angle) in a table, part of which is shown in **Table 8.5**.

If these values are now stored in a memory device such as a ROM (read-only memory) or an EPROM (erasable programmable read-only memory), the sine of any angle can be accessed if we know which address it is stored in. For simplicity, assume the address of each value is equal to the sine of the value, eg

Address 0 contains sine(0) = 0.0
Address 45 contains sine(45) = 0.707106781 etc

If the address lines of our ROM are now connected to a counter that counts from 0 to 359, each value in the table from 0 to 359 degrees can be stepped through in sequence. If we then connect a DAC to the output of the ROM, and feed that into a Low Pass Filter (LPF), a sine wave can be obtained at its output (**Fig 8.58**).

The frequency of the sine wave will depend on the speed at which the address counter is incremented. For example, if it is assumed that a 36MHz crystal clock is used to clock the address counter, then the frequency of the sine wave will be:

36MHz/360 = 100kHz.

The purity of this sine wave will depend on:

- The accuracy of the sine data stored in the ROM table;
- The accuracy of the 36MHz clock;
- The stability of the 36MHz clock; and
- The accuracy of the DAC.

If we wanted to produce a different frequency, say 200kHz, there are two options - either increase the clock rate to 72MHz or change the address counter so that it increments by two at each clock pulse. The ROM output now skips alternate addresses, so the complete 0 - 360 count is completed in half the time, ie the output frequency is doubled.

It is generally more convenient to alter the address increments rather than the clock frequency each time it is desired to change frequency. Following the above example, if the address increments by three at each clock pulse, the output frequency becomes 300kHz. The limit is reached when the address is incremented by 180 on each clock pulse, in which case the output frequency becomes (a square wave at) 18MHz.

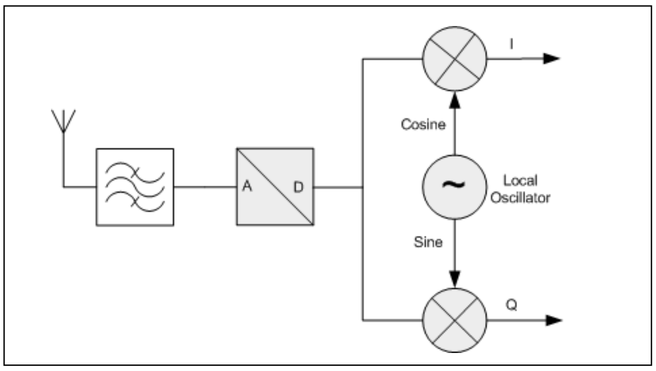


Fig 8.57: Generating the I and Q signals

Address	Angle	sine (Angle)
0	0	0
1	1	0.017452406
2	2	0.034899497
3	3	0.052335956
357	357	-0.052335956
358	358	-0.034899497
359	359	-0.017452406

Table 8.5: Part of the contents of a sine table

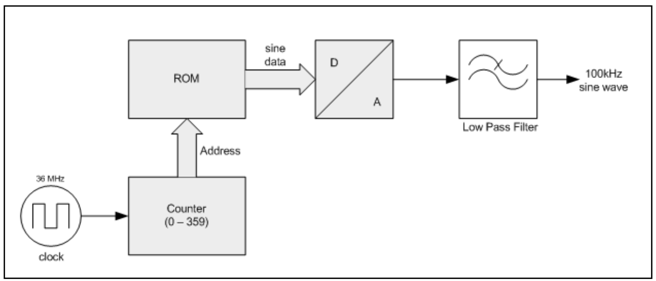


Fig 8.58: Block diagram of Direct Digital Synthesiser (DDS)

Also note that as we increase the frequency, the number of samples that is used to reproduce the sine wave is reduced.

If the DDS output is to be used as the local oscillator for a receiver, then 100kHz steps are likely to be far too coarse, as most modern radios tune in 1Hz or less steps.

In order to provide such fine steps, a much larger ROM is used, which contains finer angular resolution steps. For example, let's assume our clock remains at 36MHz and it is desired to tune in 1Hz steps. In this case, we will need to store the sine(angle) at:

1Hz/36MHz = 0.000000027 degree steps

Hence for 360 degrees, a ROM is needed with:
360/0.000000027 = 129.86 x 108 locations

DDS are available in integrated circuit form - the Analog Devices AD9951 is popular with amateur radio designers and experimenters. This device uses a ROM with 232 addresses, hence when clocked at, say, 500MHz the step size becomes:

500MHz/232 = 0.011Hz

This resolution should be fine enough resolution for most applications!

In practice, it is possible to get away with a quarter of the number of addresses, since we only need to store the first quadrant (0 to 90°) of our sine wave. The other quadrants can be calculated by clocking backwards for 91 to 180°, clocking forwards but inverting the sign of the ROM data (ie multiplying by -1) for 181 to 270° and by clocking backwards and inverting for 271 to 359°.

Whilst this sounds fine in theory, in practice the issues identified above that affect the accuracy of our sine wave result in unwanted spurs being present in the output waveform. Fig 8.59 shows the spurs produced by an early DDS chip - the AD9951 whilst Fig 8.60 shows the significantly better performance of later chips - an AD9912 in this case.

These spurs appear as low-level, unwanted signals as the receiver is tuned across a band of frequencies. Note that the level and number of spurs tends to increase as the local oscillator frequency is increased.

Even so, the latest DDS chips are significantly better than the first generation and, with a suitable pre-amp on the higher HF bands to raise the band noise above the spur level, are acceptable for the local oscillator of a high performance receiver.

To date, DDS techniques have typically been used to mix the incoming RF signal to a lower frequency so that a low-cost PC sound card can be used as an ADC and a PC provides the DSP function. This technique is used by the Flex-Radio SDR-1000 and FLEX-5000 SDRs, the latter having the sound card built in.

Preliminary performance of Mercury

Table 8.6 shows some performance figures measured on the final prototype for the first commercially available Mercury system. As you can see, the figures are very promising!

Unlike a conventional analogue radio, a Digital Down Conversion radio such as the Mercury receiver will have a sudden overload point, as the Analogue-to-Digital Converter (ADC) in it reaches full scale. This overload point is the limiting factor in terms of performance.

In the case of the ADC used in Mercury, the LT2208 is a saturating ADC so increasing the input signal beyond the overload point does not produce a sudden catastrophic degradation in performance.

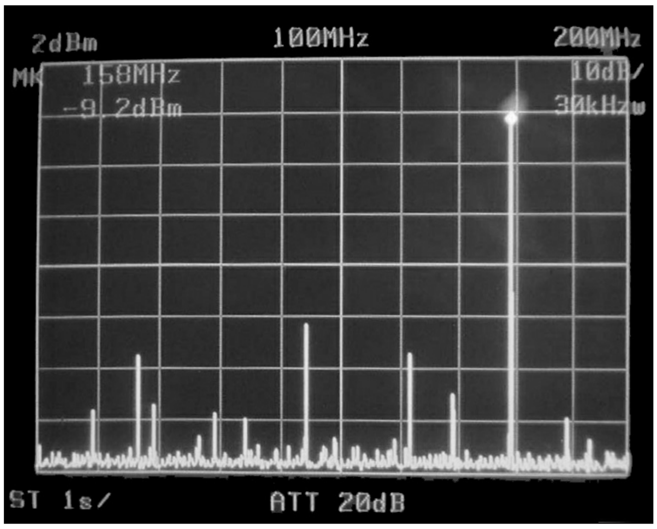


Fig 8.59: Spurs produced by an AD9951 when operating at 158MHz using a 500MHz clock (photo by Carmignani Giuliano, I0CG)

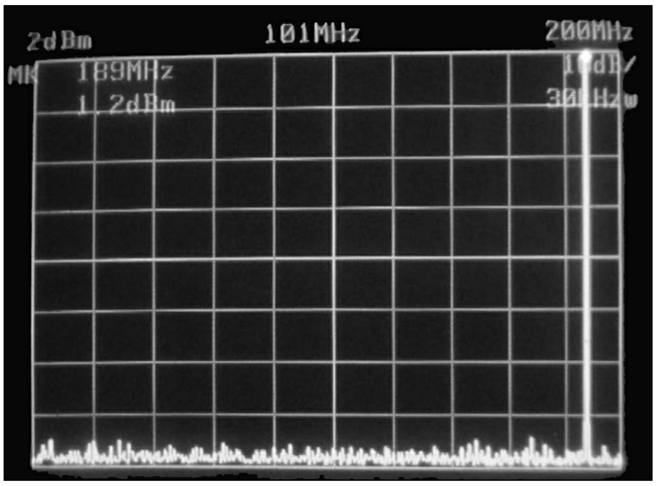


Fig 8.60: Spurs produced by an AD9912 when operating at 189MHz using a 1GHz clock (photo by Carmignani Giuliano I0CG)

Mercury prototype	Pre-amp off	Pre-amp on
Minimum Discernible Signal (MDS):	-117dBm	-135dBm
Third order intercept point (IP3):	+50dBm	+30dBm
Blocking Dynamic Range (BDR):	125dB	125dB
Overload:	+8dBm	-10dBm

The IP3 at +50dBm is approximate since it is at the limits of VK6APH's test equipment. The MDS was measured in a 500Hz bandwidth. Note that the IP3 is independent of tone spacing.

BDR was measured at 100kHz and 5 kHz for 1dB gain compression with similar results. Note that the BDR is set by the overload point of the ADC rather than being phase noise limited.

Table 8.6: Measured performance of Mercury prototypes

The phase noise of the 125MHz clock has been estimated at -149dBc/Hz, which VK6APH thinks is very good for a packaged crystal oscillator. Note that the 125MHz oscillator was not phase locked to a 10MHz reference for these tests.

The preamp used by VK6APH consisted of a noiseless feedback power FET with 2dB noise figure and 8dB of gain, driving a Norton BJT stage with 12dB of gain. ZL3IX used a two stage Norton BJT preamp.

It was noted the MDS with the pre-amp off is a little high - most receivers give about -127dBm - so a small pre-amp gain may be needed at all times to reach this level.

The figures are very good for a wide band receiver. In particular, the 5kHz BDR compares favourably with that published by the ARRL [50] when testing the ICOM IC-7800 (where a 5kHz BDR of 107dB was obtained). Although the 100kHz BDR (135dB) of the IC-7800 is better than that obtained with Mercury (125dB) the addition of a tracking RF preselector in front of Mercury (as used by the IC7-800) would substantially improve this figure.

IMPLEMENTING DDC IN A FIELD PROGRAMMABLE GATE ARRAY

Whilst there are several chips that provide either a partial or a complete Digital Down Conversion (DDC) function, most experimenters are tending to use Field Programmable Gate Arrays since this provides tremendous flexibility as to exactly how the DDC is implemented and potential for much experimentation.

The architecture to implement a DDC in an FPGA is illustrated in **Fig 8.61**. This shows the block diagram of the Verilog code used to implement the HPSDR Mercury DDC receiver in an FPGA. Note that the CIC and CFIR filters shown in the diagram were described earlier in this chapter.

It is possible to implement a DDS local oscillator in an FPGA, given one with sufficient RAM or Flash PROM to hold the sine table. However, this would require a large - and hence expensive - FPGA, or external memory.

An alternative technique, called CORDIC (which is an acronym for COordinate Rotation DIgital Computer), is more frequently used. This technique was developed many years ago to enable early electronic calculators to generate trigonometric identities. This class of algorithm is often referred to as a Numerically Controlled Oscillator (NCO).

The CORDIC algorithm basically calculates the sine and cosine of a given angle 'on-the-fly' rather than require these to be pre-calculated and held in some form of read-only memory (ROM).

The algorithm can be simply implemented in an FPGA and does not require any multiplier functions which, as previously discussed, are in rather short supply in FPGAs.

In addition, if the ADC data and desired frequency is fed into the CORDIC function, the I and Q outputs can be directly produced without the need for additional multipliers. Yet one more advantage of the CORDIC is that the multiplication by sine and cosine to produce the I and Q outputs is done without the need to convert to an analogue waveform. This means that the errors caused by non-linearity in a DAC, which result in spurs, are avoided.

However, unfortunately, this does not mean that the CORDIC algorithm is without its problems. It is necessary to generate sine and cosine of the required angle sufficiently accurately to maintain the level of spurs at an acceptable level - which is not an easy task.

With the accuracy of the sine and cosine necessary to keep the spurs down, the amount of room in the FPGA that the CORDIC code takes can become excessive. As a result of the increased size of this code, timing errors make it more difficult

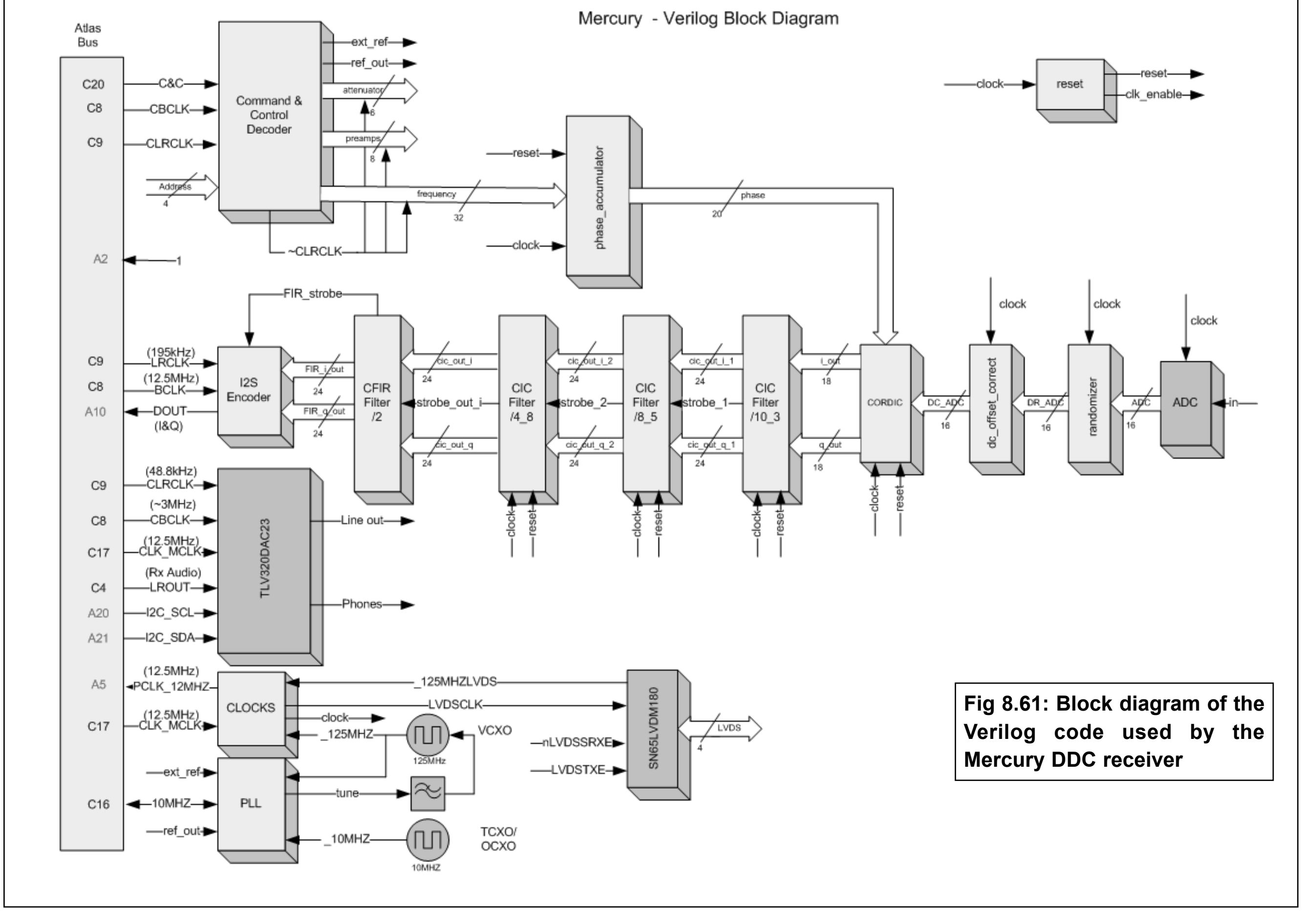


Fig 8.61: Block diagram of the Verilog code used by the Mercury DDC receiver

The CORDIC Algorithm

The CORDIC algorithm used by the HPSDR Mercury DDC receiver and Penelope DUC (Digital Up Converter) transmitter is based on an iterative algorithm.

The advantage of the algorithm is that the sine and cosine of an angle is calculated using simple shifts and addition, rather than by multiplication. This greatly simplifies the implementation in an FPGA.

Consider the point (X, Y) in **Fig 8.63.** Suppose it is desired to rotate this by an angle Z. The coordinates for the new point (Xnew, Ynew) are:

$$Xnew = X.\cos(Z) - Y.\sin(Z)$$
$$Ynew = Y.\cos(Z) + X.\sin(Z)$$

This can be re-written as:

$$Xnew/\cos(Z) = X - Y.\tan(Z)$$
$$Ynew/\cos(Z) = Y + X.\tan(Z)$$

If the angle is broken into smaller pieces so that their tangents are always a power of two (eg $\tan^{-1}(1/2^n)$) the equation can be rewritten as follows:

$$X(n+1) = P(n).(X(n) - Y(n)/2^n)$$
$$Y(n+1) = P(n).(Y(n) - X(n)/2^n)$$
$$Z(n) = \tan^{-1}(1/2^n)$$

The angle, $\tan^{-1}(1/2^n)$, needs to be pre-calculated and results in values of 45°, 26.565°, 14.0362° . . . etc.

P(n) is called the aggregate constant and can be calculated by multiplying all the P(n)s together as follows:

$$P = \cos(\tan^{-1}(1/2^0)) . \cos(\tan^{-1}(1/2^1)) . \cos(\tan^{-1}(1/2^2)) \ldots\ldots \cos(\tan^{-1}(1/2^n))$$

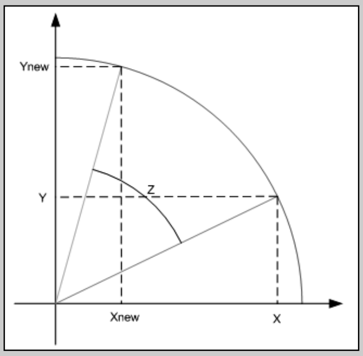


Fig 8.62: Vector rotation using the CORDIC algorithm

This is a constant equal to 0.607253 - we can either start with a vector of this length, rather than 1, or multiply the sine and cosine values generated by the CORDIC algorithm by P at the end. For use in SDRs, normally P would not be used, the then effective gain of the CORDIC stage either being used or removed by truncation.

Ignoring P, the final equations are:

$$Xnew = \text{sum}(X(n) - Y(n)/2^n)$$
$$Ynew = \text{sum}(Y(n) + X(n)/2^n)$$

The $Y(n)/2^n$ and $X(n)/2^n$ terms can be simply implemented by shifting the data one bit to the right. In pseudo code, the algorithm looks as follows:

```
For i = 1 to n-1
    If (Z(n) >= 0) then
        $X(n + 1) = X(n) - (Yn/2^n)$
        $Y(n + 1) = Y(n) + (Xn/2^n)$
        $Z(n + 1) = Z(n) - \tan\text{-}1(1/2^i)$
    Else
        $X(n + 1) = X(n) + (Yn/2^n)$
        $Y(n + 1) = Y(n) - (Xn/2^n)$
        $Z(n + 1) = Z(n) + \tan\text{-}1(1/2^i)$
    End if
End For
```

An excellent explanation of the CORDIC algorithm can be found at: http://www.dspguru.com/info/faqs/cordic.htm. This includes the CORDIC algorithm implemented using an Excel spreadsheet, as well as in the C language.

to operate the CORDIC at the high end of the local oscillator frequency range.

Despite these issues, it is possible to produce a very effective CORDIC NCO that, by using a 125MHz clock, will step in fractions of a Hz between 10kHz and 55MHz, with all spurs greater than 100dB down on the fundamental signal.

As with the DDS explained previously, the quality of the signal produced by the NCO is dependent on the stability and phase noise of the clock used to drive it. High stability and low phase noise VHF crystal oscillators are used for the NCO clock in high quality receivers to yield the best performance.

DIGITAL UP CONVERSION SDR TX - THE HPSDR PENELOPE EXCITER

Whilst much of this chapter so far has been specifically about SDR receivers, this does not mean that the transmitter side is being ignored by radio amateurs interested in SDR.

As has been mentioned earlier, the HPSDR project [8] has a digital transmitter project named 'Penelope' as part of it.

Penelope is a Digital Up Converter (DUC) exciter that generates about 0.5W of RF output over the amateur bands from 1.8MHz through to 52MHz.

The basic block diagram of a DUC-based exciter is shown in **Fig 8.63**. As old hands will observe, this is basically a phasing-type transmitter, but since the more conventional phase shifts in the audio and RF sections are done digitally, the levels of carrier and sideband suppression achieved are substantially higher than those attainable in an analogue phasing exciter.

The process starts in Penelope by digitising the microphone signal. As long as we sample at more than twice the highest modulating frequency, say 8ksps, then the exact sampling frequency is not important. However, it is useful to sample at a rate whereby a standard PC sound card can be used to listen to the digitised audio, so for the HPSDR project a 48kHz sampling rate at 16 bits is used.

The digitised microphone signal is passed to two digital bandpass filters. These set the ultimate bandwidth of the RF signal typically 300Hz to 2.4kHz for an SSB signal. The actual bandwidth can usually be set by the user, since it is under software control.

One of the outputs from the bandpass filters is passed through an additional stage called a 'Hilbert transform'. This stage shifts the phase of all signals passing through it by 90°.

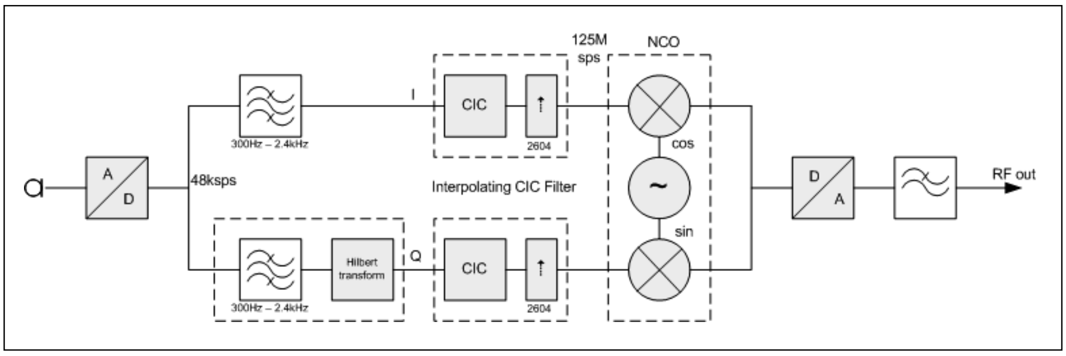


Fig 8.63: Block diagram of a Digital Up Conversion (DUC) exciter

Although the Hilbert transform is shown as a separate block in Fig 8.68, in practice it is often included as part of the bandpass filter.

At this stage, there are two digital data streams - both at 48ksps and 16bits - that can be considered as equivalent to the I and Q signals of a conventional analogue phasing exciter.

We now need to increase the effective sampling rate of the I and Q signals in order to generate an RF signal. Sampling theory requires that the I and Q signals must be sampled at least twice the highest frequency that it is desired to generate. If it is assumed that that the 6m amateur band (i.e. 52MHz) is the highest required frequency, then we need to sample at greater than 104MHz. A sample rate of 125MHz was chosen for the HPSDR project in order to simplify filtering of 'alias' products when operating on 52MHz.

The question then is: "How do we increase the sample rate from 48ksps to 125Msps?" This is achieved by using a similar technique to that used in the Mercury receiver, where the sampling rate is reduced from 125Msps to 320ksps. In the case of the Mercury receiver, every 320th sample was simply removed. For the Penelope DUC transmitter, we add - or 'bit-stuff' as it is sometimes called - additional zeros to make up for the missing samples.

In this case, it is desired to interpolate from 48ksps to 125Msps - which is a speed increase of 125,000/48 = 2,604 times. In which case 2,603 zero samples are added, each of 16 bits, following each real 48ksps sample to make up the 2,604 samples required.

This 'bit-stuffing' significantly increases the bandwidth of the original I and Q signals. As a result, before we finally mix to our desired RF frequency the new 125Msps data stream needs to be filtered - which can be done in a similar manner to the Mercury receiver.

As explained earlier, a class of digital filter called a Cascaded Integrator Comb (CIC) is a very efficient way of filtering a digital signal, particularly when the filter is required to be implemented in a Field Programmable Gate Array (FPGA).

Just as in the case of the Mercury receiver, we are able to combine the function of interpolation and filtering in the one stage - which is achieved by using an 'interpolating CIC' filter.

Again, as previously explained, the CIC filter has deep nulls at multiples of its clock/division rate - 48kHz in this case. The filter also has a 'droop' in its passband, which in the case of the Mercury receiver was corrected with the use of a CFIR filter. In the case of the Penelope transmitter, since the droop is less than 1dB at 5kHz for AM, FM and SSB modes, this does not necessarily need correcting.

The I and Q signals, suitably interpolated and filtered, now need converting to the desired RF frequency. Just as in the Mercury receiver, we could convert each of these signals into an analogue signal and apply them to two mixers and a local oscillator.

However, wishing to remain digital as close to the antenna as possible, it was decided to do the frequency conversion digitally, using multipliers for the mixers and a Numerical Controlled Oscillator (NCO) for the equivalent local oscillator.

Similarly to the case of the Mercury receiver, a Direct Digital Synthesiser (DDS) could have been chosen for this purpose, but it was decided to follow the CORDIC NCO route.

Not only is the CORDIC route very economical, since no multipliers are required to implement the algorithm in an FPGA, but it also completely eliminates the unwanted sideband that would normally require nulling using a conventional analogue design. This feature is explained in more detail in the nearby box 'Multiplying complex sinusoids'.

The output of the CORDIC NCO contains just the wanted sideband, which is converted to an analogue RF waveform using a Digital to Analogue Converter (DAC) chip.

The level of RF here is in the order of a few milliwatts and conventional analogue techniques are used to amplify up to the desired power level.

The overall block diagram of the HPSDR Penelope transmitter is shown in **Fig 8.64**. Here the Penelope DUC is connected to an HPSDR Atlas backplane together with an Ozy PC interface board.

Penelope contains an ADC for the microphone signal, the output of which is passed to the Atlas bus. An Ozy board collects these samples and passes them, via a USB 2 interface, to the host PC.

Software on the PC (eg *PowerSDR*^{TM} [12] provides the necessary bandpass filtering, phase shift and compression, etc and passes the resulting I and Q signals back over the USB link to Ozy. Ozy in turn decodes these signals and places them on the Atlas bus. Penelope receives the I and Q signals and applies them to the CIC filter, as described previously in this section.

An overall block diagram of the Verilog code used to implement the Penelope DUC transmitter in an Altera EP2C8 FPGA is shown in **Fig 8.65**. Like all the code used in the HPSDR project, this is open source and can be downloaded using SVN - see the HPSDR [8] web site for further details.

Multiplying complex sinusoids

A complex sinusoid can be represented by the expression

$$\exp(j\omega) = \cos(\omega) + j\sin(\omega)$$

If two complex sinusoids, f1 and f2, are multiplied together then the result, Z, is given by

$$\begin{aligned} Z &= \exp(jf1) \times \exp(jf2) = \exp(j(f1+f2)) \\ &= \cos(f1 + f2) + j\sin(f1 + f2) \end{aligned}$$

which contains only the sum (f1 + f2) and not the difference frequency (f1 – f2).

PROTECTING SDR 'FRONT-ENDS' - THE ALEXAIRES HPF/LPF SYSTEM

There are some areas of leading-edge receiver and transmitter design where both analogue and SDR radios share similar problems. In analogue receivers, the combination of narrow multi-pole crystal roofing filters and fast bus switches (such as the famous FST3125 and the later FSA3157) as mixers has resulted in designs of the latter providing third order intercept points (IP3) of 40 to 50dBm.

Similarly, in SDR digital down conversion receiver boards such as the HPSDR's Mercury the use of high performance analogue-to-digital converters (ADCs) like the Linear Technologies LT2208 has meant that ADC overload levels of +10dBm can be obtained.

What this means is the one-time weak-point of both types of radio is now very strong - and that the limitation of the strong signal performance of both types of receiver is more often determined now by design of the RF preselector/high pass filtering which precedes the mixer (in analogue radios) or ADC (in SDRs).

Receiver high pass filters (HPF) - or to use their traditional name, RF preselectors - have been traditionally designed to keep out-of-band signals out of the input of a receiver. Over the last four decades, their design has been fairly straightforward, as it was relatively easy task to produce one that did not affect

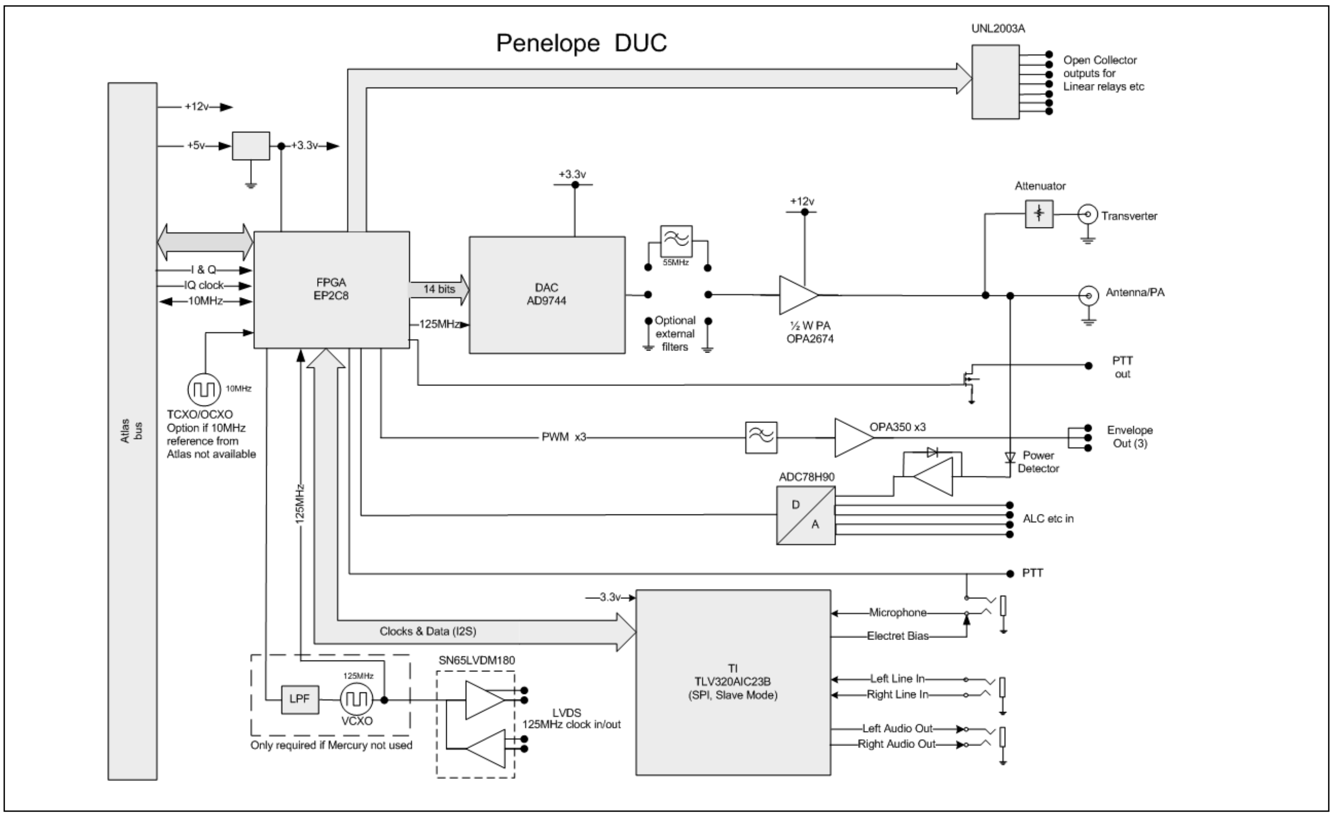


Fig 8.64: Block diagram of the HPSDR 'Penelope' DUC

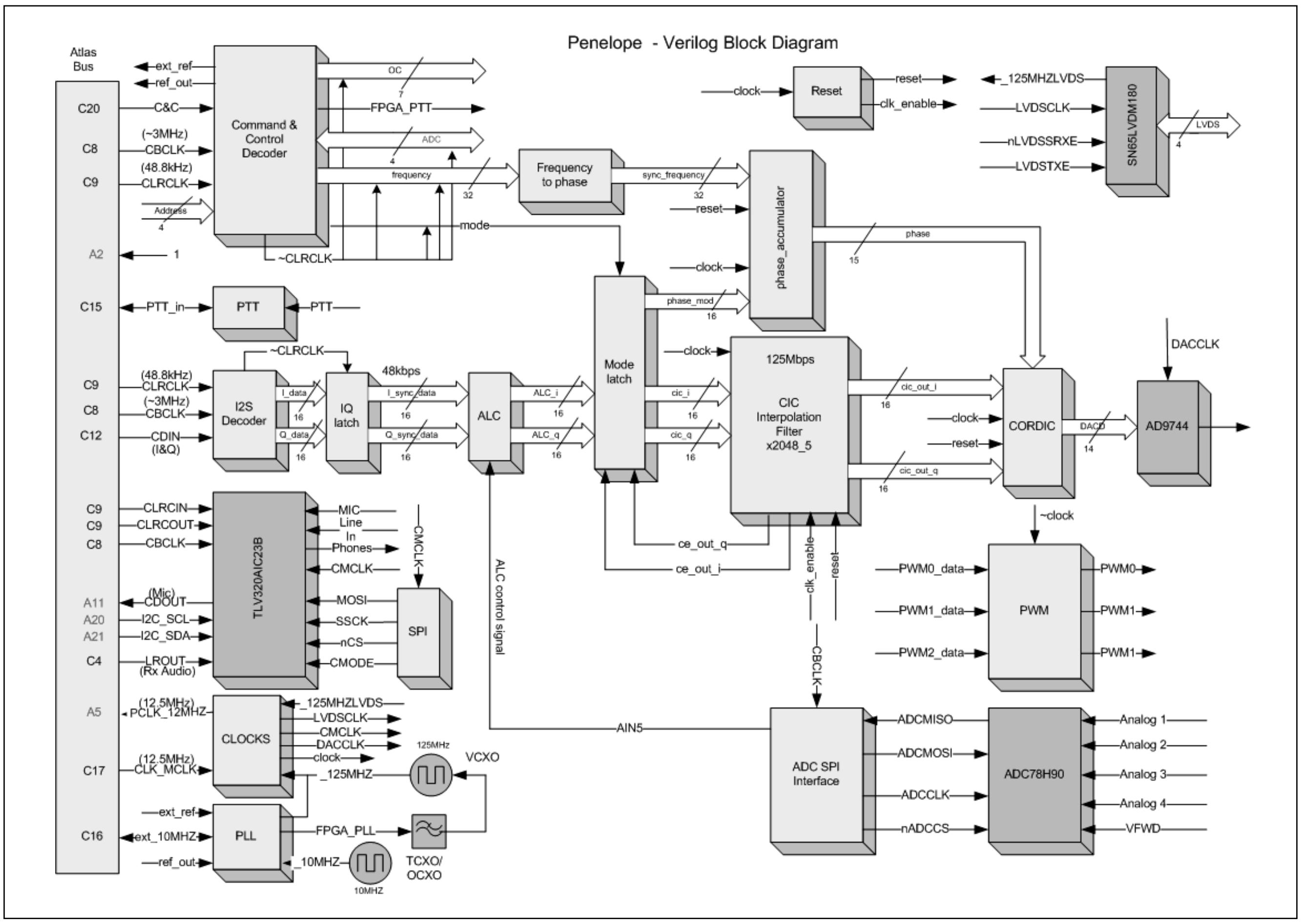


Fig 8.65: Block diagram of the Verilog code used to implement a DUC

the relatively poor IP3 performance of the first mixer of most receivers/transceivers.

About 18 months ago, VK6APH started some design experiments with Graham Haddock, KE9H, to produce an RF preselector for the HPSDR. This project was given the name of Alexaires - 'Alex' for short - named after one of the Greek gods of defence. **Fig 8.66** shows a block diagram.

As a receiver preselector, the purpose of 'Alex' is not only to reduce the level of out-of-band signals at the receive input of the HPSDR's Mercury receiver board, but, equally importantly, to suppress any signals at the images or alias frequencies that appear at multiples of its sampling clock frequency/2 (122.8MHz/2) plus and minus the operating frequency.

'Alex' also has a built-in transmitter low pass filtering (LPF) and will suppress the harmonic energy typically generated by a 100W RF power amplifier. The bank of transmit low pass filters (LPFs) are also used to provide additional input band limiting for Mercury.

When VK6APH was testing out some experimental prototypes of Alex using off-the-shelf inductors, he quickly discovered that for receivers such as Mercury with a potential equivalent IP3 of +50dBm you need to be very careful as to the choice of inductors used at the front-end if one is not to significantly degrade the IP3 performance. VK6APH found that the problem lay with the toroidal cores used in the preselector inductors, which, although relatively linear at low flux values, quickly became non-linear as the flux values increased. Air cored inductors were perfectly linear, but getting a high-enough Q from them was a problem - as was their size and unwanted ability to couple with each other.

In a bid to solve this problem, VK6APH tried increasing the size of the core being used, going as far as using the T200-2 specification of toroidal core, which is five centimetres across and more usually used in HF transmitting baluns. Even with a core of this size and type there was loss of flux linearity at very high signal levels, apparently resulting in a very slight degradation of IP3.

At this point, VK6APH started to suspect his own test equipment and do some serious searching on the Internet. He found some fascinating information on the website [51] of Martein Bakker, PA3AKE, a well-known experimenter in leading-edge analogue HF receivers, who had run into exactly the same problem with an RF preselector he had built to go in front of a H-mode mixer. PA3AKE was helped in his analysis by Colin Horrabin, G3SBI, who was one of the designers for the CDG2000 transceiver which appeared in *RadCom* a few years back [41].

A few days later, VK6APH found himself at the Dayton Hamvention and happened across the German Hilberling PT8000A high-performance receiver, which actually used T200-2 cores in its preselector -see **Fig 8.67**. It seemed that everyone had come across the same problem. At this point, VK6APH and KE9H started working very seriously on a final design for Alex.

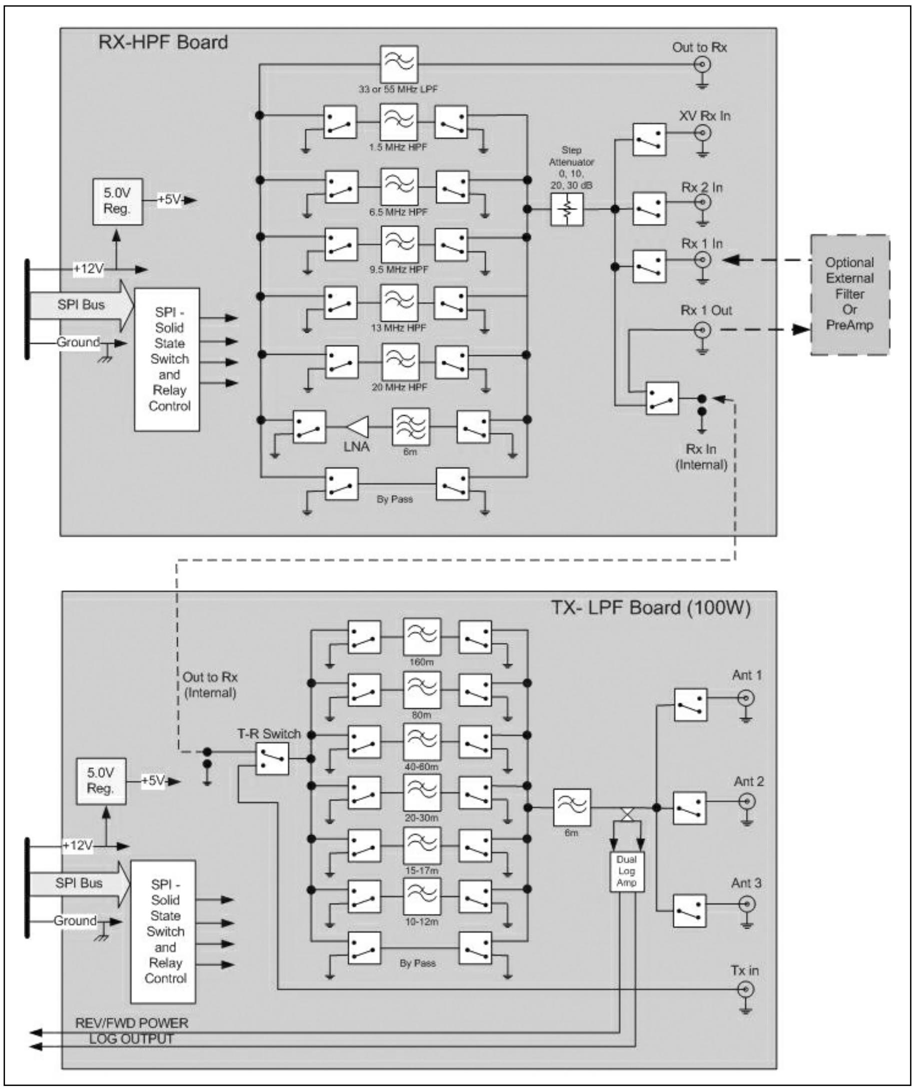


Fig 8.66: Block diagram of the Alexaires (Alex) preselector / filter system

As outlined earlier, there are at least three major issues to consider with a preselector for a DDC SDR receiver such as Mercury - and first is protecting the receiver from overload. The LT2208 ADC used in Mercury is highly linear and robust, but it does have limits. The maximum input level that the LT2208 converter can accept before its overall performance becomes degraded is 2.25V peak-to-peak or +11dBm, which is S9 plus 84dB.

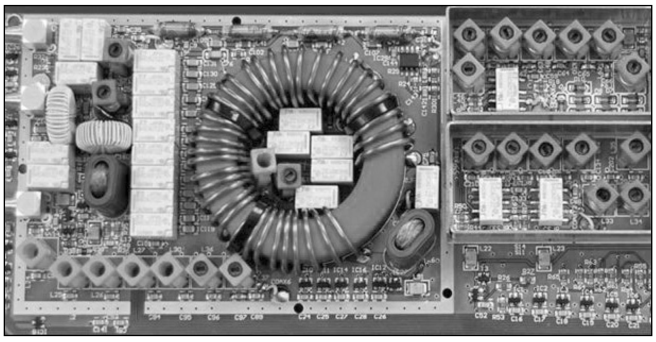


Fig 8.67: Hilberling PT8000A preselector board, showing its T200-2 core

A local broadcast transmitter or very local amateur transmitter might generate this kind of level. Signals of this level do not need to be eliminated - just reduced so that the ADC is not overloaded.

The second issue is that of linearity, which has been detailed earlier. The native linearity of the LT2208 ADC is so high that it is difficult to measure with traditional test equipment and the potential limitation on receiver performance is the non-linearity of toroidal inductors - and electronic switches - in a multi-band RF preselector that precedes it.

The large-signal performance of an SDR receiver using the LT2208 will be degraded due to the gain of any preamplifier that is used (typically 10 to 20 dB with an LT2208 if you wish to use it on all the amateur bands from 1.8 to 54MHz), so a preselector IP3 target performance of around +40 dBm was used.

The third is image suppression. As mentioned earlier, the Mercury receiver uses a sampling frequency of 122.88MHz for the ADC. This means it will directly sample signals in the spectrum from 0 to 61.44MHz band, but signals above those frequencies may also appear in the data converter output. For example, assume that a signal at 10MHz is desired to be received, so a signal at 132.88MHz and a signal at 112.88MHz will appear as 'aliases' unless they are prevented from getting to the ADC input

Of more practical concern are the VHF TV and FM signals that could 'fold back' (ie alias) into the 1.8 to 54MHz amateur bands if not eliminated. For example, the 88 to 108MHz FM band could appear as images from 15 to 35MHz, while the US TV Channel 4 audio sub-carrier at 71.75MHz will fold back to 51.13 MHz.

As the ADC has no native selectivity or rejection at alias frequencies, the total system selectivity of the antenna system, matching networks and preselector frequency rejection must add up to about 120 to 140dB, depending on how strong VHF TV and FM signals are in your area. This also has implications for requirements for shielding the SDR ADC and the RF preselector filters.

The HPSDR Alex consists of two Euroboard-sized printed circuit boards (10 cm by 16 cm), intended to be mounted in a commercial Euroboard extruded aluminum housing, such as a Hammond 1455N1601. It does not plug into the HPSDR Atlas bus, but is intended for separate mounting and controlled by an SPI bus, provided via an IDE connector on Mercury.

The first PCB is the receiver filter board and consists, firstly, of a 33 or 55MHz 7th-order LPF that is always in line with the receiver. If the user is going to operate on the 50MHz amateur band the 55MHz version should be used, otherwise the 33MHz version is recommended (for better VHF image suppression).

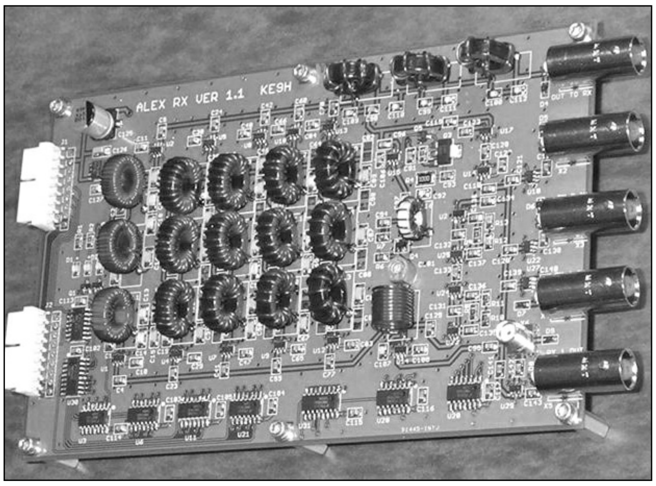


Fig 8.68: HPSDR Alex receiver filter board

The receiver filter board (Fig 8.68) also contains a 6m low-noise preamplifier; selectable 20MHz, 13MHz, 9.5MHz, 6.5MHz and 1.6MHz high pass filters (HPFs), plus a switchable 'front-end' attenuator, configurable for 0, 10, 20 or 30dB attenuation. It has five external BNC connectors for interconnections and the selection of three different receive antennas in addition to the transmit antenna.

The second PC board is the transmitter filter board and consists of seven relay-switched LPFs for transmitter harmonic suppression, used in conjunction with the receiver HPFs to provide a flexible variable bandwidth receiver input filter function. It has four external BNC connectors for interconnections and the selection of three transmit antennas.

Each active board contains three LEDs, so that an indication of power, transmit/receive switching and SPI bus operation can be seen.

In terms of specifications for Alex, its insertion loss is variable according to frequency, but typically will not exceed 2dB total for receive paths, and 0.5dB for transmit paths. The transmit harmonic filter banks and associated relay switches are intended to handle up to 100 watts with a CW or SSB duty cycle. Its total power consumption should be less than 1.5W.

So, getting back to where we started, what is Alex's contribution to receiver IP3 performance? VK6APH and KE9H are glad to say that in the Alpha versions it was measured at approximately +50dBm - that is to say that there is no measurable degradation of Mercury's native IP3 performance.

NEED FOR AN SDR 'CRYSTAL SET' TO TEACH DSP

For the experienced radio amateur who enjoys ‘home brewing’, there are numerous opportunities to experiment with new Software Defined Radio ideas and techniques – in both hardware and software.

For the hardware enthusiast, we have already seen that very high performance SDRs can be built at a low cost, and have a low level of complexity when it comes to their components – for example, the SoftRock range of kits [2] and the High Performance Software Defined Radio (HPSDR) range of ready-made boards [8].

This trend is likely to continue in the future since, as faster signal processing devices become available, more and more of the SDR hardware will be consumed/replaced by software.

For the programmer who wants to play around with SDRs, there are a number of options, depending on your skill level. For the relative beginner, the planned changes to the open source *PowerSDR™* code [12], which will result in the separation of the Digital Signal Processing (DSP) part of the software from the Graphical User Interface (GUI), will provide the opportunity for them develop their own GUI for *PowerSDR™*, in the programming language of their choice.

For the more experienced programmer, working in the bowels of DttSP [25] - the software that commonly implements the core DSP functions of a Software Defined Radio (SDR) - can be a rewarding experience.

There is little doubt that having a basic understanding of the concepts behind DSP is a valuable asset as you progress along the path to understanding SDR.

However, the main issue facing most radio amateurs is how to make the transition from understanding analogue radio to that of understanding SDRs.

Back in the old days, if you were a beginner in (analogue) radio, the usual starting point for your education was to build a crystal set. This ‘hands-on’ approach also introduced many new practical skills – such as soldering, fabrication of parts, etc - that could be built-on, as your interest and understanding of radio grew.

The problem with understanding SDR technology, in particular DSP, is that there is no equivalent teaching aid to the 'crystal set'. To the beginner, or even a seasoned analogue engineer for that matter, most DSP text books seem to be to be written in a foreign language and appear mostly unintelligible.

Below is an attempt to provide a newcomer to SDR and DSP techniques with some practical experience in actually processing signals digitally, so they can feel confident about understanding the brave new digital world - and perhaps, ultimately, even a DSP text book!

So, if we consider that a soldering iron is the main tool needed to build our first crystal set, what is its software equivalent? There are many excellent tools available for simulating DSP systems, but since most are aimed at the professional engineer they carry a hefty price tag, together with a significant learning curve.

One afternoon, on a long boring plane flight to Tokyo, VK6APH came up with the idea of using simple, low-cost, spreadsheets – *Microsoft™ Excel™* or its Open Office [52] equivalent – as the software equivalent to the soldering iron when it came to the digital world.

After some hours of experimentation, he found that many fundamental DSP principles can be easily explained using a spreadsheet. In fact, as we will see a little later in this section, quite advanced DSP processes can be both modelled and designed using this tool.

Perhaps the major advantage of using *Excel™* in this way is that many radio amateurs that do not have a technical background are familiar with its use - in fact, many will use it each day in their work for financial calculations, etc.

INTRODUCTION TO DIGITAL SIGNAL PROCESSING

Analogue to Digital, Digital to Analogue

In order to use digital signal processing (DSP) the real world analogue signals must be converted to digital (A-D) and back again (D-A). A detailed description of these processes can be found in the Principles chapter.

DSP Learning Laboratory

Whilst there are several excellent DSP teaching tools currently available, these tend to be expensive for the beginner. As a result, instead of using one of these, the following examples are based on the use of a spreadsheet.

In order to conduct a number of DSP experiments, it is first necessary to develop some items of test equipment. Using a spreadsheet, the following will be developed:

- An Analogue-to-Digital Converter;
- A Digital-to-Analogue Converter;
- An Oscilloscope;
- A Signal Generator, providing CW, AM and FM modulation modes, together with a sweep function and in-phase (I) and quadrature (Q) outputs;
- A Digital Filter; and
- A Spectrum Analyser

A signal generator

In order to test and evaluate the building blocks that are used in DSP, a signal source is needed. Whereas in the analogue world, either an off-the-air signal or a signal generator would be used, in the DSP/SDR world either a fixed frequency or a noise source is used instead.

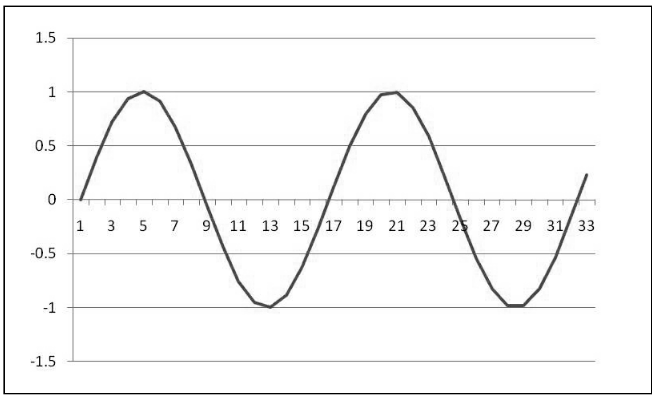


Fig 8.69: Sine wave from a signal generator

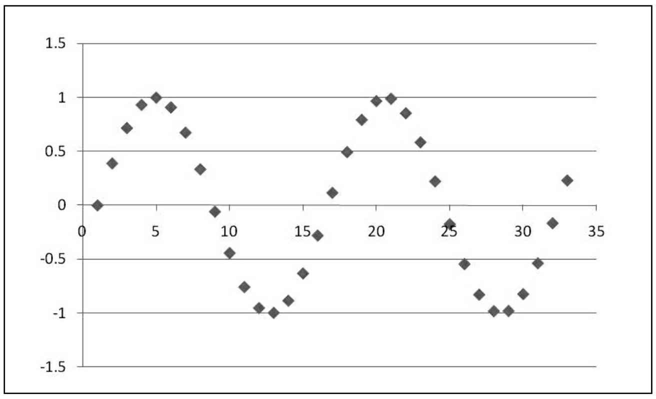


Fig 8.70: Sampled sine wave

Actually, what we are going to do in this case is to simulate generating a test signal in a very simple manner using a spreadsheet - which will enable us to explain some aspects of digital 'sampling' at the same time.

In the analogue world, a signal generator can produce a sine wave output at a specific frequency and amplitude – as is shown in **Fig 8.69**.

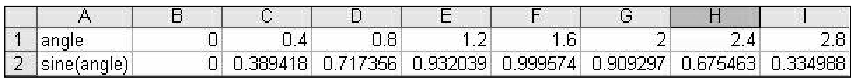
In the digital world, rather than generating a continuous signal, a sine wave is produced which is sampled at regular intervals – see **Fig 8.70**. Each of these samples is held in a 'cell' within a spreadsheet – see **Fig 8.71**. It is convenient therefore to think of each spreadsheet cell as representing a signal sample value.

To generate a sampled sine wave, we could calculate the value of the wave at a fixed incremental angle and enter these into the spreadsheet. However, there is a much easier way to generate this sine wave - let the spreadsheet software do all the calculations for us!

To create a sine wave using a spreadsheet, we simply enter the line below into a cell:

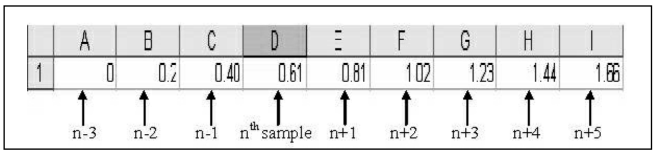
= sine(value)
where 'value' is in radians and 1 radian = 180/π degrees

If we gradually increase the 'value' in incremental steps, entering each increment into a separate cell, a sampled sine



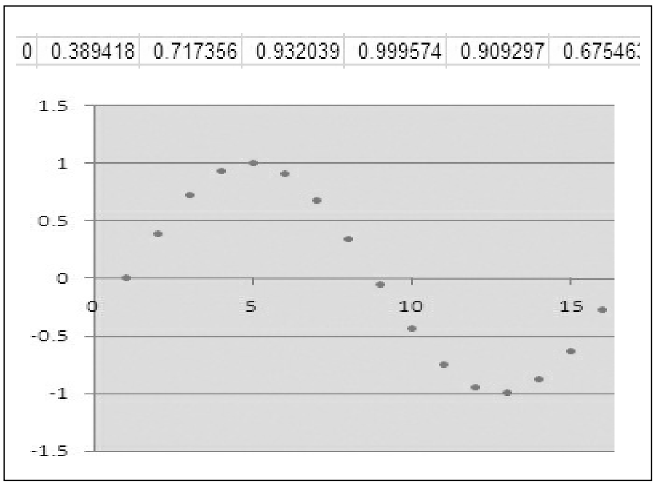
	A	B	C	D	E	F	G	H	I
1	angle	0	0.4	0.8	1.2	1.6	2	2.4	2.8
2	sine(angle)	0	0.389418	0.717356	0.932039	0.999574	0.909297	0.675463	0.334988

Fig 8.71: Samples held in a spreadsheet



	A	B	C	D	E	F	G	H	I
1	0	0.2	0.40	0.61	0.81	1.02	1.23	1.44	1.66

Fig 8.72: Sample identification



0	0.389418	0.717356	0.932039	0.999574	0.909297	0.67546:

Fig 8.73: Sine wave graph produced by spreadsheet. At the top is part of the series used to plot the points on the graph

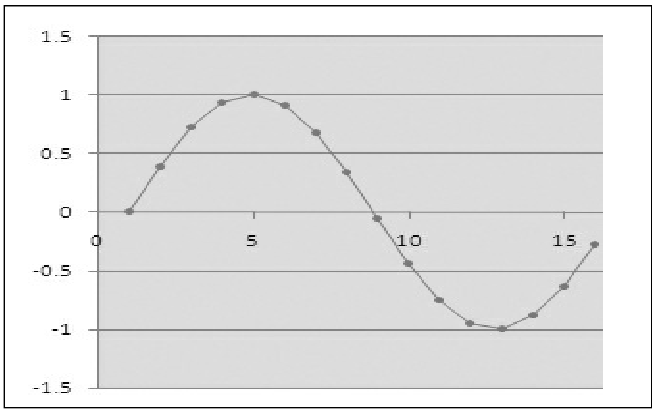


Fig 8.74: Using a spreadsheet to perform an analogue to digital conversion

wave is produced. The more cells we enter data into, the longer (in time) the sine wave will exist.

It is very important to understand the concepts of 'current', 'previous' and 'future' samples, so we will examine these in more detail. First, let us consider the row of cells in the spreadsheet, shown in Fig 8.72.

If, say, cell D1 is selected and called 'n', then the cell immediately to the left (cell C1) will be n-1, and the cell to the left of that (cell B1) will be n-2, and so on.

Similarly the cell to the immediate right of D1 (cell E1) will be n+1 and to the right of that (cell F1) n+2.

The amplitude of the sine wave can be plotted using *Excel's* graphing tools, resulting in Fig 8.73. *Note: This* Excel *file (sine.xls) can be found on the disc that accompanies this handbook.*

The sine wave consists of a series of discrete samples, rather than a continuous waveform. In order to convert these samples into an analogue signal, a digital-to-analogue conversion needs to be performed.

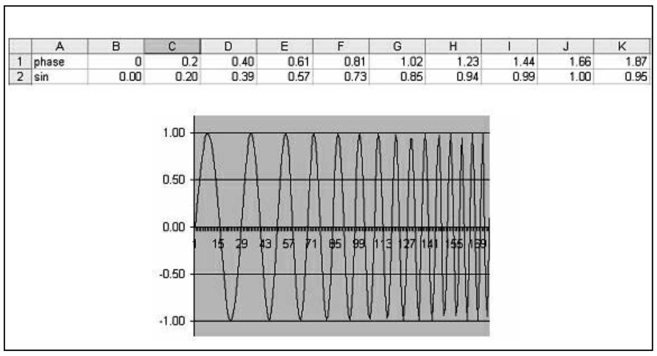
This can be done by simply joining the sample dots together with straight lines. Fortunately, Excel's graphing tools allow us to do this with a simple click of the mouse and the result is shown in Fig 8.74.

Whilst a fixed-frequency signal generator is useful, what would be even more useful is one that would sweep over a frequency range. This way, it could be connected to the input of a digital circuit and we could plot the frequency response of the latter.

Producing a frequency sweep is again very easy in *Excel^{TM}* - rather than incrementing 'value' with equal amounts, we simply add progressively larger values with each sample, for example:

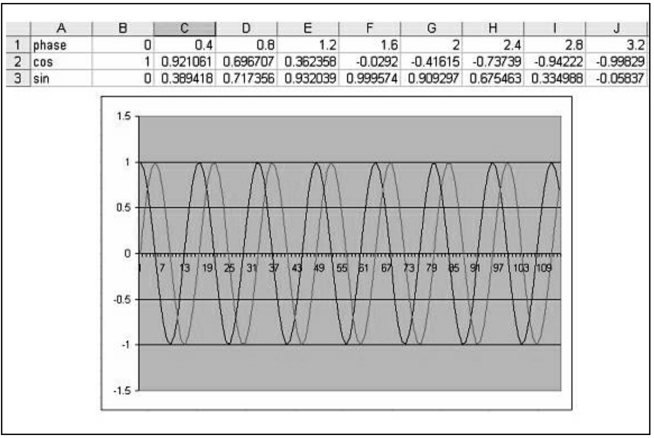
sine[value + (previous value * x)]

This is shown in Fig 8.75 *and is available in the file sweep.xls, which can be found on the compact disc which accompanies this handbook.*



	A	B	C	D	E	F	G	H	I	J	K
1	phase	0	0.2	0.40	0.61	0.81	1.02	1.23	1.44	1.66	1.87
2	sin	0.00	0.20	0.39	0.57	0.73	0.85	0.94	0.99	1.00	0.95

Fig 8.75: Sweep generator



	A	B	C	D	E	F	G	H	I	J
1	phase	0	0.4	0.8	1.2	1.6	2	2.4	2.8	3.2
2	cos	1	0.921061	0.696707	0.362358	-0.0292	-0.41615	-0.73739	-0.94222	-0.99829
3	sin	0	0.389418	0.717356	0.932039	0.999574	0.909297	0.675463	0.334988	-0.05837

Fig 8.76: I and Q waveforms

In-phase (I) and Quadrature (Q) signals

The importance - and concept - of I (in phase) and Q (quadrature) signals for Software Defined Radio (SDR) has already been explained earlier in this chapter.

Since processing I & Q signals is essential for SDR, it's really useful to be able to simulate/generate them using *Excel^{TM}* – which is relatively simple to do.

If you recall, I and Q signals are displaced in phase from each other by 90° or $\pi/2$ radians. If we consider that the sine wave generated in Fig 8.75 is the I signal, then to create the Q signal we simply add $\pi/2$ to each value, ie

Q = SIN(value + $\pi/2$)

The result of this is shown in Fig 8.76. Notice that both sine waves have the same frequency and amplitude, but are 90° out of phase with one another. Another way of thinking about these two signals is that one is a sine wave and the other a cosine wave. In fact, the Q signal could have been generated in *Excel*™ using:

Q = COS(value)

See the file I&Q.xls, which can be found on the compact disc which accompanies this handbook.

Amplitude modulation

Excel™ can be used very effectively generate and demodulate an Amplitude Modulated (AM) signal.

An AM wave can be described by multiplying the carrier sine wave by a modulating signal. Let's use the sine waves we have previously generated as our carrier and create another sine wave as our modulation. Since our modulation is usually at a lower frequency than our carrier, let's use 1/5th of the carrier frequency for our modulating signal.

This means our modulation becomes:

Modulation = SIN(value/5)

To create AM, the modulation is multiplied by the carrier, so in *Excel* we use:

AM = (0.5 + M*SIN(Modulation) * SIN(Carrier)
where M is the modulation depth (0 – 100%)

If this is plotted using *Excel's* graphing facility, we see the familiar AM waveform - **Fig 8.77** *(see the file AM.xls, which can be found on the compact disc which accompanies this handbook).*

Since we would like to demodulate the AM signal from the I & Q signals we need to also generate a Q signal with the same format. We do this the same way but this time we use:

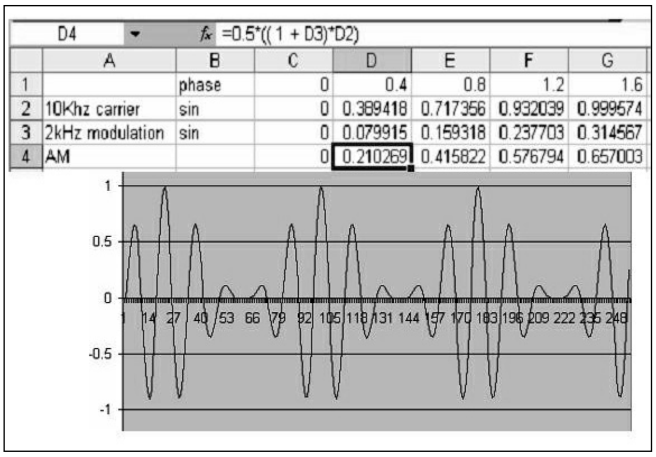
AM_Q = (0.5 + M*SIN(Modulation) * COS(Carrier)

Again we can plot the I and Q signals and the result is **Fig 8.78**. Notice that both I and Q have the same form, they are just phase-shifted with respect to their carriers by 90°.

To demodulate these signals and recover the modulation we use Pythagoras' Theorem:

$$AM = \sqrt{(I^2 + Q^2)}$$

or as an *Excel*™ formula



D4 =0.5*((1 + D3)*D2)

	A	B	C	D	E	F	G
1		phase	0	0.4	0.8	1.2	1.6
2	10Khz carrier	sin	0	0.389418	0.717356	0.932039	0.999574
3	2kHz modulation	sin	0	0.079915	0.159318	0.237703	0.314567
4	AM		0	0.210269	0.415822	0.576794	0.657003

Fig 8.77: Amplitude modulation

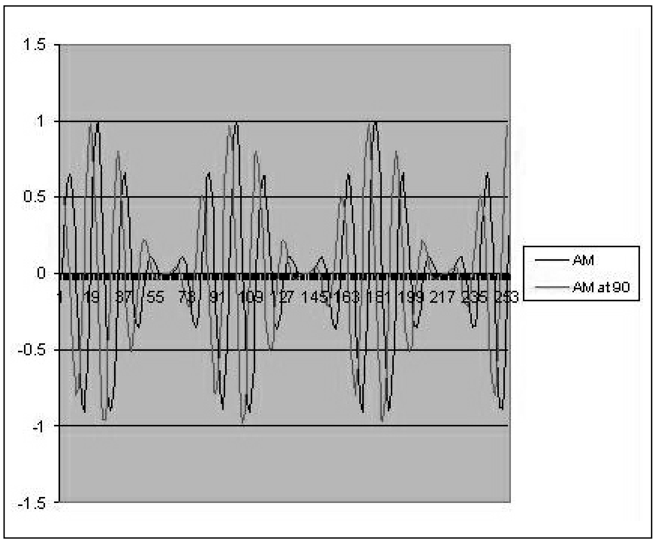


Fig 8.78: I and Q AM waveforms

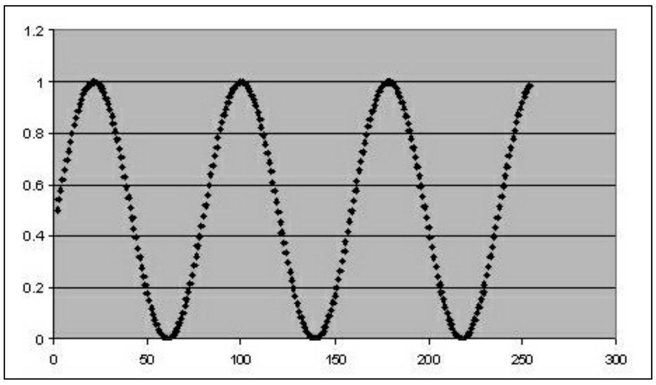


Fig 8.79: I and Q AM demodulation

= SQRT(I^2 + Q^2)

This result is shown in **Fig 8.79** which demonstrates that we have recovered the original modulation signal *(see the file AM2.xls, which can be found on the compact disc which accompanies this handbook).*

As has been explained before, this technique works for any carrier frequency, even one that has the carrier within the audio frequency range or at 0Hz.

Phase and frequency modulation

In a similar manner to AM, phase modulated (PM) and frequency modulated (FM) signals can be simulated and demodulated.

An FM signal can be generated in a similar manner to our AM test waveform. In this case:

FMI = SIN*(Carrier + Modulation)

and FMQ = COS*(Carrier + Modulation)

I and Q signals can be used to demodulate these waveforms as a frequency modulated signal, using:

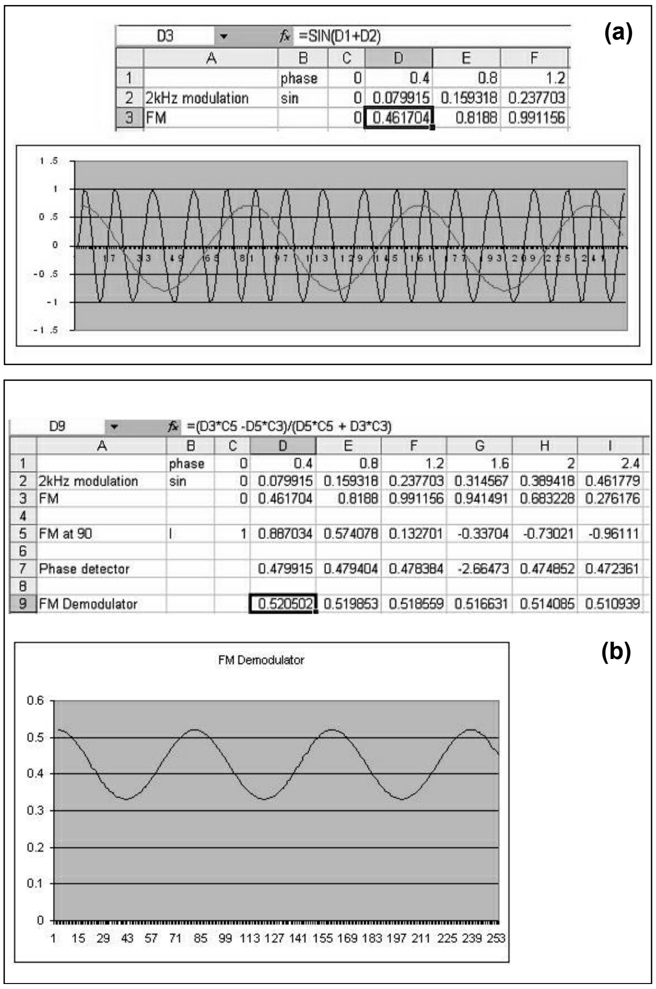
$$FM = (Q_n . I_{n-1} - I_n . Q_{n-1})/(I_n^2 + Q_n^2)$$

where n is the current sample and n-1 is the previous sample. The result of performing this process on the I and Q signals is shown in **Fig 8.80** *(see the file FM.xls, which can be found on the compact disc which accompanies this handbook).*

A couple of things should be noted about this technique. Firstly, since frequency modulation is the rate of change of phase, it can be seen that the process involves subtracting the previous samples from the current samples. In effect, this gives us the slope of the signal, which is the same as the rate of change of phase.

Secondly, the resulting signal is determined by the ratio of the I and Q signals and not their absolute values. Hence the recovered modulation will remain constant over a wide range of signal levels and any AM noise or interference will not appear on the output signal.

This is similar to the effect we get when receiving an FM signal on a conventional analogue radio, where the signal has passed through a limiter.



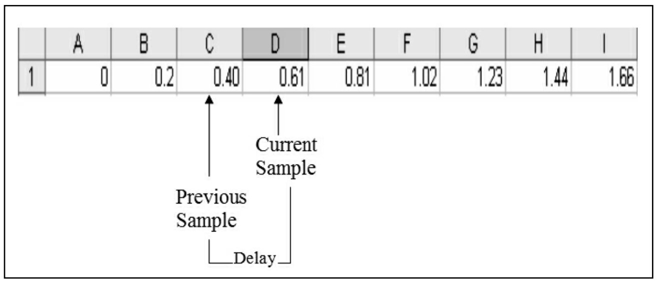
D3 =SIN(D1+D2)

	A	B	C	D	E	F
1		phase	0	0.4	0.8	1.2
2	2kHz modulation	sin	0	0.079915	0.159318	0.237703
3	FM		0	0.461704	0.8188	0.991156

D9 =(D3*C5 -D5*C3)/(D5*C5 + D3*C3)

	A	B	C	D	E	F	G	H	I
1		phase	0	0.4	0.8	1.2	1.6	2	2.4
2	2kHz modulation	sin	0	0.079915	0.159318	0.237703	0.314567	0.389418	0.461779
3	FM		0	0.461704	0.8188	0.991156	0.941491	0.683228	0.276176
4									
5	FM at 90	I	1	0.887034	0.574078	0.132701	-0.33704	-0.73021	-0.96111
6									
7	Phase detector			0.479915	0.479404	0.478384	-2.66473	0.474852	0.472361
8									
9	FM Demodulator			0.520502	0.519853	0.518559	0.516631	0.514085	0.510939

Fig 8.80: (a) FM generation and (b) FM demodulation



	A	B	C	D	E	F	G	H	I
1	0	0.2	0.40	0.61	0.81	1.02	1.23	1.44	1.66

Fig 8.81: Spreadsheet samples

Digital filters

For those of us like VK6APH and VK6VZ who have been brought up in an analogue world, understanding the concept of being able to filter a signal digitally takes a little patience. After all, we know that resistors, capacitors and inductors can be used to create all sorts of filters and have been using them for this purpose for years and, well, they just work!

So how can we take a series of signal samples and filter them, just like we have done for years in the analogue world?

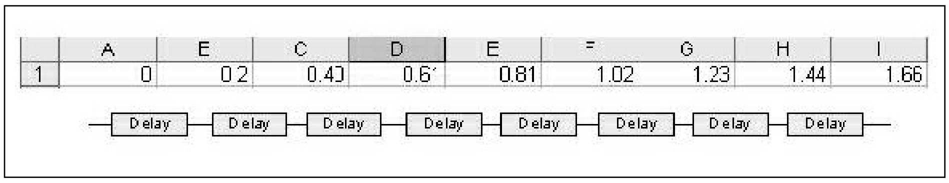
Well it turns out that not only can all the filters used in the analogue world be reproduced digitally, (eg low-pass, high-pass, band-pass) but when implemented digitally they often have significantly better performance.

By better, we mean:

- Digital filters are always 100 per cent reproducible, since there are no hardware component tolerances to worry about.
- Digital filters do not alter their characteristics with time, temperature, humidity, shock or vibration.
- The performance of digital filters can be accurately predicted and modelled.
- There is no need to 'tweak' or align digital filters.
- Any changes to digital filters are made by changing software, not component values, so different filters - or variable filters - can be implemented easily. These changes can be made whilst the filter software is running, ie 'adaptive' filters can be produced.
- Compared with, say, a bank of narrow-band, high-performance CW or SSB filters, digital filters are low cost.
- The performance of digital filters (eg shape factor) can exceed that of conventional analogue filters.
- Digital filters can be implemented that would be either impossible or very expensive to produce using analogue techniques.

This doesn't mean that an analogue filter can always be replaced with a digital one, since before a digital filter can be used on a signal, the latter needs to have first been digitized. Even then, a digital signal needs to be at a suitable sample rate that currently available digital signal processors can handle. That being said, the fact is that digital filters are replacing analogue ones at a rapid rate.

The fundamental building block of a digital filter is the delay element. This element simply stores a sample for a clock cycle and then passes it on. Now, we can think of each cell in our



	A	B	C	D	E	F	G	H	I
1	0	0.2	0.40	0.61	0.81	1.02	1.23	1.44	1.66

Fig 8.82: Cascaded delay element

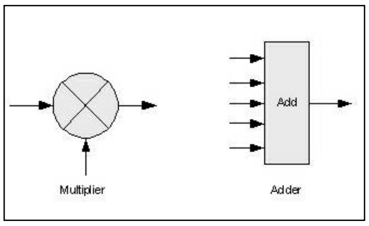


Fig 8.83: Adder and multiplier

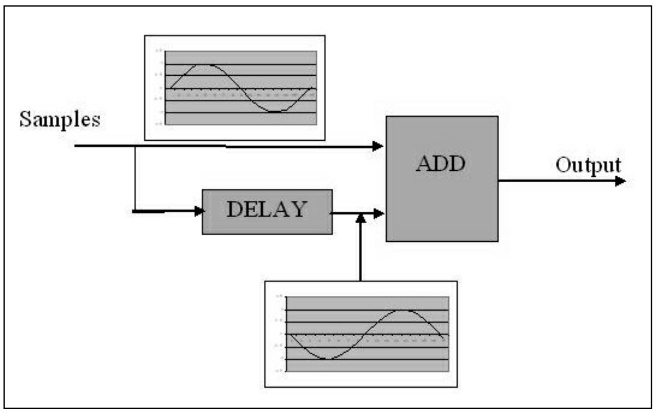


Fig 8.84: Comb filter

Excel™ spreadsheet as a delay element. For example, if the samples in **Fig 8.81** are available in our spread sheet, we could consider them as delay elements, as shown in **Fig 8.82**.

In order to create a digital filter we need to introduce two other elements, the adder and multiplier - see **Fig 8.83** These elements operate much as expected - the output of the adder is simply the sum of its inputs, while the output of the multiplier is the multiplication of its inputs.

In *Excel™* we can use:

Output = A + B

for an adder, and:

Output = A*B

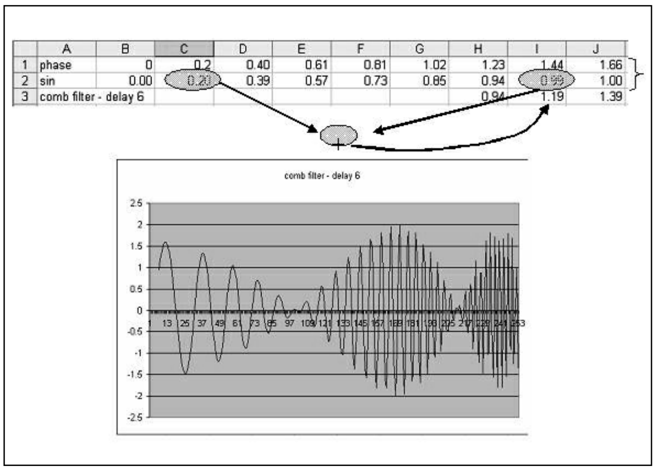
for a multiplier.

The simplest digital filter that can be made is a comb filter, so-called because of the shape of its frequency response. A comb filter can be made from a delay element and an adder, as per **Fig 8.84**. This can easily be implemented in *Excel™*, as follows:

Output = Current Cell + Previous cell

For explanations sake, we could assume that the input signal is a 1kHz sine wave and the delay is 0.5mS. In which case the output of the delay element will be exactly 180° out of phase with the other input to the adder. As a result, adding these two signals will result in zero output. The output will also be zero for all harmonics of 1kHz, ie 2, 3, 4 kHz, etc.

If our sweep generator is applied to the input of the filter and the output plotted again using *Excel™*, the results can be



	A	B	C	D	E	F	G	H	I	J
1	phase	0	0.2	0.40	0.61	0.81	1.02	1.23	1.44	1.66
2	sin	0.00	0.20	0.39	0.57	0.73	0.85	0.94	0.99	1.00
3	comb filter - delay 6							0.94	1.19	1.39

Fig 8.85: Comb filter frequency response

shown as the graph in **Fig 8.85** *(see the file comb.xls, which can be found on the compact disc which accompanies this handbook).*

The frequency response has recurring notches that repeat at multiples of the signal sampling rate. As you can see, the response has a comb-like structure, hence the filter's name.

The location of the filter nulls depends on the length of the total delay - the plot shown in Fig 8.81 uses a delay of six cells.

At first sight, the usefulness of a comb filter may not be apparent - after all, it's not the sort of filter you would want to pass 14MHz SSB signals through! But what if you had a signal that contained significant 50Hz mains 'hum'? Typically, such hum comprises the 50Hz fundamental plus numerous harmonics, eg 100Hz, 150Hz, 200Hz, etc.

A suitably designed comb filter would remove the fundamental 50Hz signal plus its entire harmonics and have relatively little effect on other frequencies that are passing through it.

Admittedly, such a filter would also be relatively easy to implement using analogue components; a simple switched capacitor filter would do the trick.

Comb filters are used extensively in Digital Up/Down Converters (see earlier) since they provide a high performance without the need for multipliers.

Finite impulse response filters

The Finite Impulse Response Filter (FIR) is a very popular digital filter that is inherently stable (which is more than can be said for a number of analogue filters that VK6APH has built over the years!). The 'finite' in its name comes from the fact that the filter only uses a finite number of input samples in order to produce its output.

A simple FIR digital filter can be evaluated that has a frequency response that is similar to the simple low-pass RC network shown in **Fig 8.86**. The way the various digital elements are connected is shown in **Fig 8.87**.

In addition to the delay and components of our comb filter, we have added a multiplier with a coefficient of a_n. In the FIR filter shown, we set:

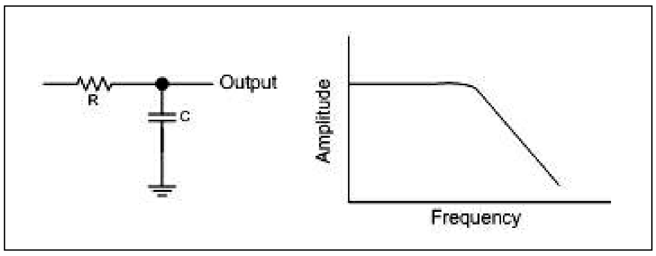


Fig 8.86: Analogue low pass filter

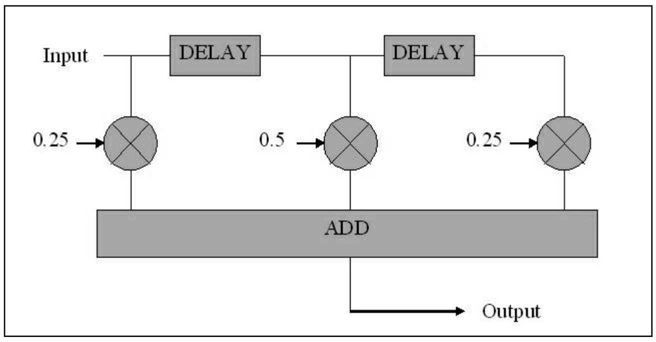
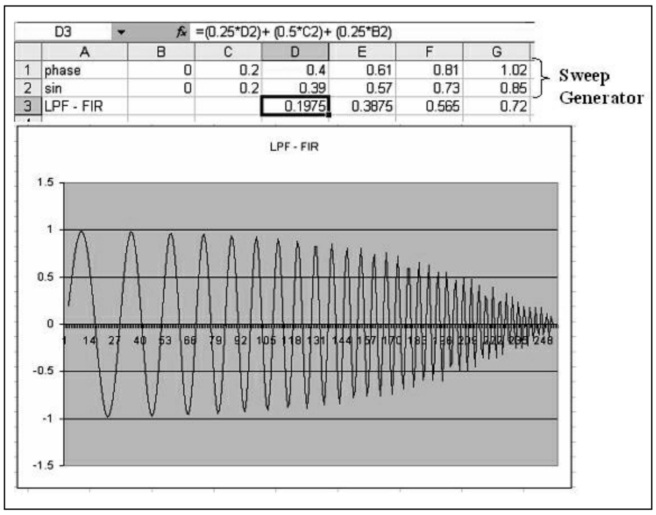


Fig 8.87: Digital low pass filter


D3 =(0.25*D2)+ (0.5*C2)+ (0.25*B2)

	A	B	C	D	E	F	G
1	phase	0	0.2	0.4	0.61	0.81	1.02
2	sin	0	0.2	0.39	0.57	0.73	0.85
3	LPF - FIR			0.1975	0.3875	0.565	0.72

Fig 8.88: Excel implementation of a low pass filter and the resultant graph

$a_0 = 0.25$
$a_1 = 0.5$
$a_2 = 0.25$

Each cell in the filter spreadsheet will have the following format:

Output = (0.25*n) + (0.5* n_{-1}) + (0.25*n_{-2})
where n = current cell

The frequency response of this FIR filter can be plotted by feeding the input from our sine wave sweep generator. This gives the result shown in **Fig 8.88** *(see the file LPF.xls, which can be found on the compact disc which accompanies this handbook)*.

As can been seen, this looks exactly like the response we get from the analogue-equivalent RC low-pass filter.

By using more delay sections in the filter (which is called making it a 'higher order' filter), different coefficients and adding the relevant multipliers and adders, filters can be created with different characteristics, eg faster roll-off with frequency, as well as high-pass and band-pass filters.

The multiplier constants a_n can be obtained from existing design tables of digital filters.

There are also numerous software programs where you can enter the desired filter response and the program will calculate the values for you - the free ScopeFIR program [53] being a good example.

Spectrum displays

One of the most useful features of a SDR is its ability to display the spectrum of a signal, or band of signals within a region of interest. Such spectrum displays - also called bandscopes or panadapters - have been available in high-end professional and amateur radio equipment for many years. However, the use of DSP techniques in recent years for bandscopes has significantly increased their resolution, and hence usefulness, for weak signal detection/reception.

The limitation of previous analogue-based spectrum displays has been that to obtain high sensitivity and frequency resolution, a narrow filter was required. Such a narrow analogue filter was 'swept' across the frequency band of interest, but to retain the signal amplitude this needed to be done slowly, so as to prevent the filter from 'ringing'. In practice, such a slow-moving filter is unable to display CW signals in real time.

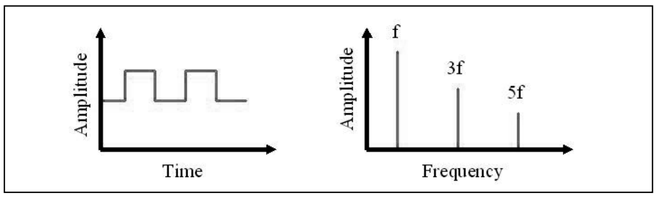


Fig 8.89: Fourier representation of a square wave

The digital solution is not to sweep a digital filter across the frequency band, since this would have the same speed limitations as an analogue filter, but rather use thousands of similar filters in parallel. Each filter can be a few Hz wide and stacked side-by-side to cover the frequency band of interest.

As a frequency band is 'swept' across, the output of each filter is selected in turn and the resulting values graphed to produce a frequency display.

Whilst it would be possible to use this multiple filter-type technique in an analogue bandscope, the cost and size of the resulting hardware would be prohibitive to say the least!

So, how are we able to implement so many filters digitally? The answer is to use a mathematical technique called Fourier analysis and in particular what is termed the Fourier Series.

In simple terms, the Fourier Series assumes that any repetitive waveform can be broken down into a series of waveforms, consisting of a fundamental sine wave plus its harmonics. Each sine wave has a particular amplitude and phase and they are all added together to give the original waveform. If we can determine what these are, then we can plot the frequency spectrum that corresponds to the waveform.

This concept is illustrated in **Fig 8.89**, where a square wave is converted to its frequency spectrum. As can be seen, the spectrum consists of a fundamental sine wave plus its odd harmonics. The amplitude of the harmonics are equal to the amplitude of the fundamental sine wave divided by the harmonic number, eg A/n.

The mathematical explanation of how the Fourier Series works is a little beyond this introductory chapter. For those who are interested, a very well written explanation can be found at [54].

The particular Fourier analysis that is desired to be performed is the conversion of the signals that we have as a series of samples into a plot of amplitude against frequency.

A Discrete Fourier Transform (DFT) is used to do this conversion. The term 'discrete' comes from the fact that the input signal consists of a number of discrete samples, rather than a continuous signal.

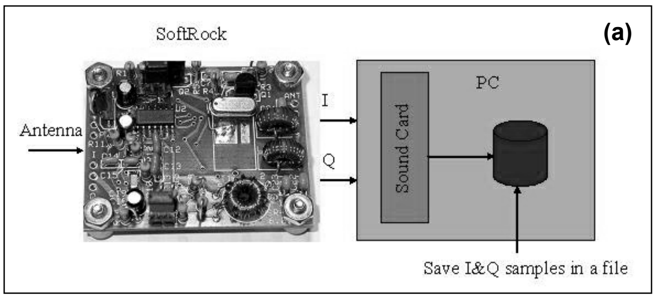
It is quite possible to implement the DFT using standard formulas in *Excel™*. In fact, for the reader that wants to get a really solid understanding of exactly how this process works, then this would be a great exercise.

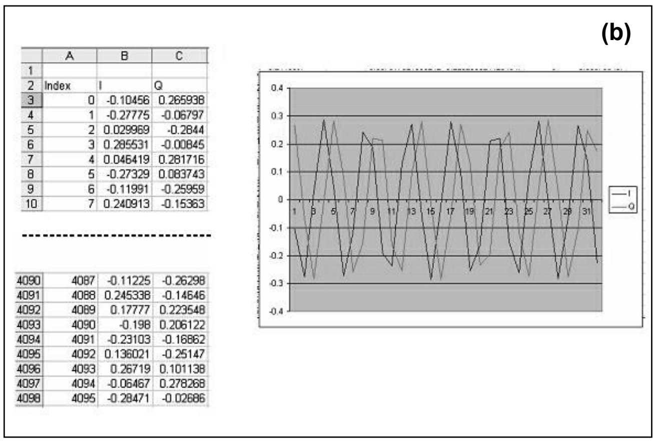
Accessing DSP analysis tools in *Excel™ 2003*

As initially installed, *Microsoft Excel™* does not provide access to a number of tools that are required to undertake the Digital Signal Processing tasks used in this chapter. To add these tools to the version of this program you have installed, select: Tools\Add-Ins\Analysis ToolPak and click OK.

To use the built-in FFT analysis tool in *Excel™*, click on 'Tools' in the menu bar and select 'Data Analysis'.

In the window that opens, select 'Fourier Analysis' and enter the range where the input samples are to be found and where the results are to be placed.





	A	B	C
1			
2	Index	I	Q
3	0	-0.10456	0.265938
4	1	-0.27775	-0.06797
5	2	0.029969	-0.2844
6	3	0.285531	-0.00845
7	4	0.046419	0.281716
8	5	-0.27329	0.083743
9	6	-0.11991	-0.25959
10	7	0.240913	-0.15363

4090	4087	-0.11225	-0.26298
4091	4088	0.245338	-0.14646
4092	4089	0.17777	0.223548
4093	4090	-0.198	0.206122
4094	4091	-0.23103	-0.16862
4095	4092	0.136021	-0.25147
4096	4093	0.26719	0.101138
4097	4094	-0.06467	0.278268
4098	4095	-0.28471	-0.02686

Fig 8.90: (a) Data capture from a SoftRock receiver. (b) SoftRock I & Q data imported into *Excel^{TM}*

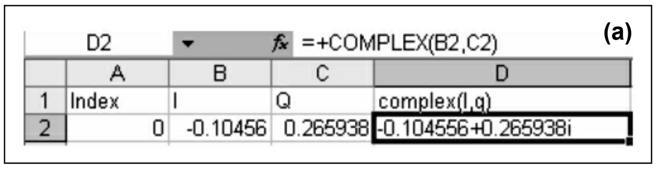
However, for those less committed - like both the authors - there is a much simpler solution. Conveniently, the DFT is such a useful analysis tool that it is already built into *Excel™*. All that is necessary is to 'point' *Excel™* at that range of sample values (which must be a power of two, eg 2^2 = 4, 2^3 = 8, 2^4 = 16, etc) and all the hard work is done for us.

Note that the Fast Fourier Transform (FFT) and a few other functions are not normally activated in *Excel™* - see the side bar *Accessing DSP analysis tools in Excel™* that explains how the DSP analysis functions we require can be added.

The use of an existing function or library is how most programmers would implement a DFT. The *PowerSDR™* software uses FFTW [55] (the Fastest Fourier Transform in the West – even mathematicians have a sense of humour it would seem!) whilst the Mercury project in HPSDR uses Ooura's FFT algorithm [56] that has been ported to C# by Phil Covington, N8VB.

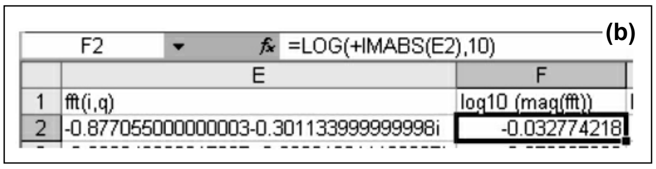
To create our bandscope display, we simply use the FFT function in *Excel™* and graph the results.

Some samples are obviously needed to feed into the FFT, so this time we will use some real data captured from the output of a sound card connected to a SoftRock receiver [2] and this has



D2 ▾ fx =+COMPLEX(B2,C2)

	A	B	C	D
1	Index	I	Q	complex(I,q)
2	0	-0.10456	0.265938	-0.104556+0.265938i



F2 ▾ fx =LOG(+IMABS(E2),10)

	E	F
1	fft(i,q)	log10 (mag(fft))
2	-0.877055000000003-0.301133999999998i	-0.032774218

Fig 8.91: (a) Converting to complex notation in *Excel* and (b) The resultant *Excel* formula

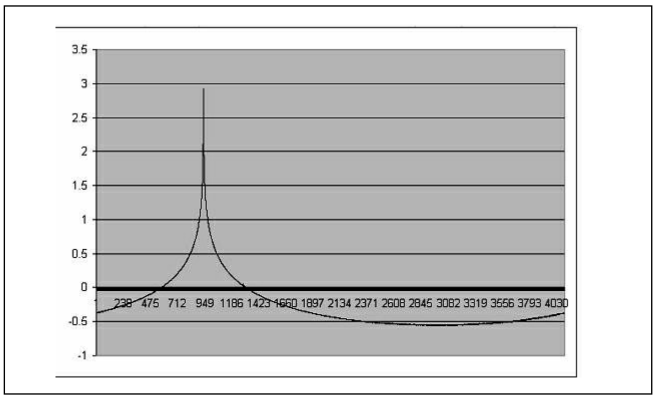


Fig 8.92: Result of using *Excel's* FFT function

been placed on the disk which accompanies this handbook *(see the file FFT.xls which can be found on the compact disc which accompanies this handbook)*. This concept is illustrated in **Fig 8.90(a)**.

This data was captured as 24-bit I and Q signals, sampled at 48kHz and consists of 4,096 samples - see **Fig 8.90(b)**. The input signal to the SoftRock was at approximately 11kHz from zero beat and about 50dB above the noise floor.

Incidentally, the data could be captured thanks to a program written by Bill Tracey, KD5TFD, which stores the data into a file suitable for directly importing into *Excel™*.

Before we can use *Excel™* to calculate the FFT, the I and Q data captured from the PC sound card needs to be converted into the required format. This is termed complex notation, where the I and Q samples are formatted as I + Qi.

Fortunately, complex notation is frequently used and a function to convert to this format is built into *Excel™* **Fig 8.91(a)**:

The FFT function in *Excel™* is then applied to the complex data which results in new I and Q terms being calculated. In order to plot the amplitude of the FFT result, we need to calculate the magnitude of the result.

We do this by taking $\sqrt{(I^2 + Q^2)}$. Again, this is a function that is built into *Excel™*. Finally we take $\log_{10}$ of the magnitude, so the amplitude can be expressed in dB. The resulting *Excel™* formula is shown in **Fig 8.91(b)**:

The result of using *Excel's* FFT function, followed by taking the log of the magnitude of the FFT, is shown in **Fig 8.92** *(see the file FFT.xls, which can be found on the compact disc which accompanies this handbook)*. Whilst this is starting to look like a signal, you would expect to see on a bandscope, something about its appearance is not quite right.

For a start there is no noise along the bottom of the display and the overall shape of the display looks odd. So what's gone wrong?

Well, in fact, nothing is wrong - *Excel^{TM}* has produced the FFT of the input samples we provided it with and correctly graphed the output. Let's look again at what our input samples look like (see **Fig 8.93**).

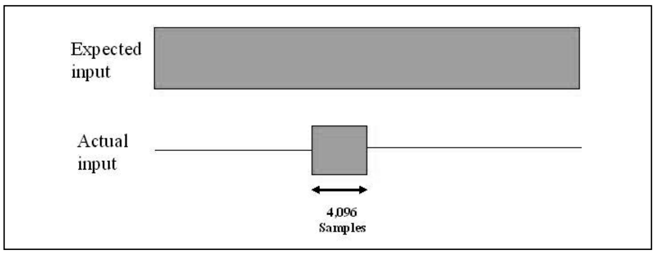


Fig 8.93: The input signal

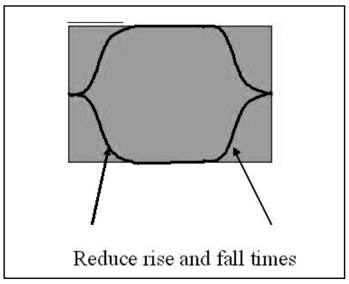


Fig 8.94: Analogue 'key-click' filtering

Table 8.7: Characteristics of several windowing functions

Window Function	Magnitude Resolution	Frequency Resolution
Rectangle	Very Low	Very High
Hanning	High	Low
Hamming	Normal	Normal
Blackman	High	Low
Blackman-Harris	Very High	Very Low
Bartlett	Low	High

The FFT is intended to be applied to a continuous stream of data, but what we have provided it with is actually a short 'burst' of data, 4,096 samples long. As a result, the input signal looks like a single CW 'dot' and, since it has a very short rise and fall time, what could be termed 'key clicks' have been generated.

The FFT spectrum that has been calculated is effectively a single CW dot with very bad key clicks, which account for the wide skirt to the signal. The sharp rise and fall times of the input signal produce numerous additional signals that add together and appear in the bandscope as an increase in the signal level along the bottom of the display.

In the analogue world, key clicks can be got rid of by passing the generated CW signal through a filter that slows down the rise and fall times before it is actually transmitted - see **Fig 8.94**. Exactly the same thing can be done in the digital world, using a technique called 'windowing'.

There are several windowing functions that have slightly different characteristics, but all tend to smooth off the leading and trailing edges of the sequence of samples that are going to be converted.

The popular windowing functions, generally named after their originators, have different advantages and disadvantages. Some of these are shown in **Table 8.7**:

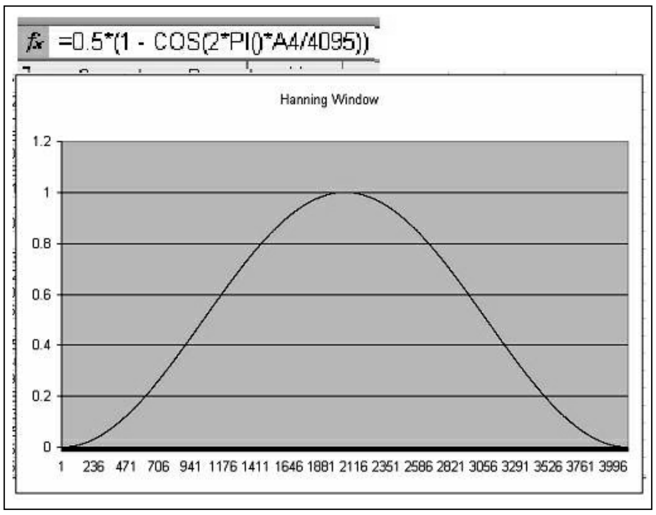

fx =0.5*(1 - COS(2*PI()*A4/4095))

Fig 8.95: Hanning window function

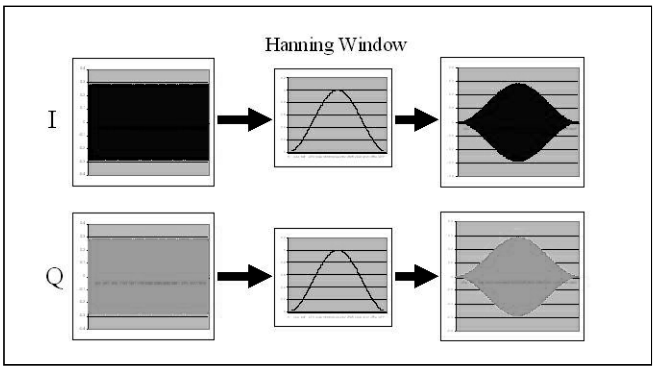


Fig 8.96: Application of Hanning window to I and Q samples

Our input data shown in Fig 8.90 has effectively been multiplied by a rectangular window having a maximum value of one.

Along with other SDR software, the *PowerSDR™* software allows the user to select from a range of different windowing functions that includes those in the table.

The Hanning window is very easy to demonstrate using *Excel™*, the formula being:

Output = 0.5 + 0.5cos(n π / (M + 1))
where n is the current sample, M is the total number of samples

If this window function is plotted in *Excel™*, then the result shown in **Fig 8.95** is achieved.

If we now multiply our I and Q samples by the Hanning window function, the rise and fall times of the signal are smoothed out and the 'key clicks' removed - see **Fig 8.86**.

If this windowed data sequence is fed into *Excel's* FFT function and the output is graphed, the image shown **Fig 8.97** results.

This looks much more like the type of signal image we would expect to see on a SDR bandscope. The vertical axis of the 'bandscope' has been modified so 10log of the signal is taken so the bandscope can be display the signal level in dB. In addition, the horizontal axis of the bandscope has been scaled in frequency – let us see how the latter has been achieved.

The 4,096 input signals (I and Q) have been sampled at 48kHz, so we also have 4,096 samples in our FFT output signal. Hence the distance between each spectrum line on the bandscope will be:

48,000/4,096 = 11.7Hz

Each of these samples is referred to as an FFT 'bin' and in this case each bin is 11.7Hz wide. Armed with this information, we can scale the x axis of the graph accordingly.

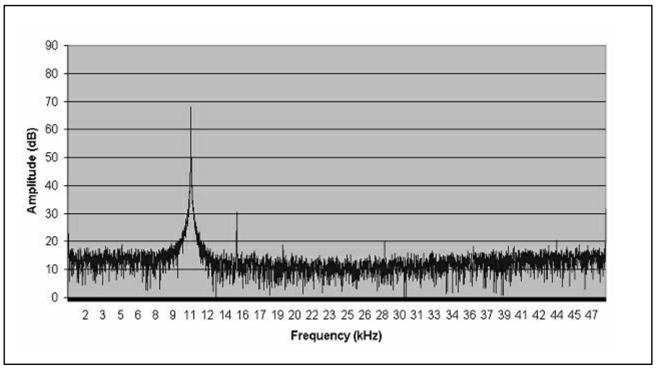


Fig 8.97: The resulting bandscope display

Conclusion

This section of the chapter introducing digital signal processing has been presented by VK6APH as a talk/demonstration at the Dayton Hamvention in May 2007, to an audience of around 350 radio amateurs. *Video of this talk is available as a digital file on the disk that accompanies this handbook*.

By using a spreadsheet - *Excel™* in this instance (although the open source 'Open Office' equivalent could also have been used instead) - the basic principles of Digital Signal Processing have been illustrated. These include:

- Signal sampling (ie analogue to digital conversion);
- Conversion of a series of digital samples back to an analogue signal (ie digital to analogue conversion);
- The concept and generation of I & Q signals;
- Digital techniques for the modulation and demodulation of AM and FM signals;
- An introduction to the Fast Fourier Transform and Spectrum Analysis;
- The process that produces a bandscope; and
- The use of windowing functions.

Although spreadsheets are used here to introduce the SDR beginner to DSP, their use is an excellent simple and low cost means of simulating - and designing - complex DSP algorithms and is also recommended to the advanced experimenter.

Further reading

- *A Simple Approach to Digital Signal Processing*, Craig Marven and Gillian Ewers, published by Wiley-Interscience.
- *The Scientists and Engineer's Guide to Digital Signal Processing*, Steven W Smith. Hardcopy or free download from: http://www.dspguide.com

A BUDGET SDR TRANSCEIVER KIT

Since the SoftRock series of Software Defined Radio kits were introduced in late 2005, several thousands of these kits have been marketed by Tony Parks, KB9YIG, and built by radio enthusiasts across the globe [2]. These simple SDR receivers, with either single or dual amateur band coverage, have been sold in the United States for between US$10 and US$20 and have performance comparable with analogue products costing several hundreds of dollars.

As more and more of these receivers were sold, Tony and an informal pool of radio designers – including VK6APH - who support the SoftRock philosophy of showing the capabilities of SDR through the sale of cheap kits to radio enthusiasts across the globe began to discuss the development of a companion transmitter or SDR transceiver for the SoftRock range.

This pool of designers includes Jan Verduyn, G0BBL, Alan Rowe, M0PUB and John Law, G8BTR. In late 2006, the three started work with KB9YIG on an SDR transmitter, which metamorphosed a few months later into the SoftRock RXTX.v6.1 SDR transceiver.

Most of the original v6.1 RXTX transceiver kits covered the 80 and 40m amateur bands, although a number covered 160m (of which VK6VZ has a prototype). The first 1,000 v6 RXTX kits sold out in around three months and a production run of a further 1,000 kits (v6.2, which mainly cover 40m/30m) started in May 2007. The transceiver produces around one watt of RF, from a pair of BS170 FETs in push-pull, and its receive section has a performance similar to that of the SoftRock v6 receiver.

A block diagram of a typical SDR transceiver is shown in **Fig 8.98**. The circuit diagram and building instructions for the RXTX transceiver can be downloaded from the Softrock 40 Internet web site [13].

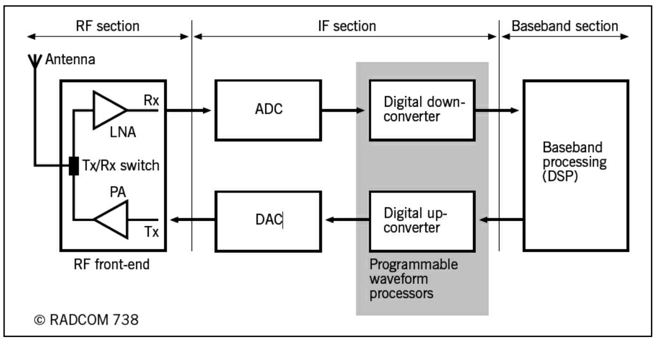


Fig 8.98: Block diagram of a typical SDR transceiver

The 160m prototype RXTX v6.1 tested by VK6APH and VK6VZ displayed a minimum detectable signal (MDS) of -115dBm and a noise figure of 32dB, using an M-Audio D44 soundcard set to -10dBm as the input signal level. On a 40m version of the RXTX v6.1, DG8SAQ measured third order intermodulation distortion on transmit at 38dB below PEP, with the harmonics greater than 50dB down – very acceptable figures [57].

If you are using a soundcard in your personal computer with a 48kHz sampling rate, the original RXTX v6.1 covers part of the 40m amateur band (in CW, digital or SSB mode) from 7.000 MHz to 7.075MHz. The coverage is extended up to 7.095kHz if you use a 96kHz soundcard, such as the M-Audio Delta 44.

This coverage is provided by means of two crystals and a jumper (or switch) is used to switch from one to another, moving the transceiver from the CW/digital modes part of 40m to the SSB portion. The kit also allows coverage of the 80m band from 3.500MHz to about 3.545MHz, but an external low-pass filter with a cut-off frequency of 4 MHz needs to be added on the RF output of the transceiver to make sure no harmonics are transmitted on the 40m band. As a result of needing this extra filtering, most of the builders of the RXTX v6.1 so far have used their transceivers exclusively on 40m.

According to G0BBL, building the RXTX transceiver kit should take from seven to eight hours upwards and be within the capability of radio amateurs who have had some previous kit building experience. There are around 25 0.1µF capacitors and seven integrated circuits in the RXTX v6.1 which are surface mounted devices ('SMDs') and require a fine tipped soldering iron, anti-static precautions, good eyesight and a steady hand.

For those who are entering the world of SoftRock construction for the first time, VK6VZ would highly recommend building a SoftRock receiver, such as the v6 lite version, before trying their hand at the RXTX v6.1 – the v6 lite receiver has less than 50 per cent of the components of the latter and is a great way to get started with the various types of SDR software that can be used with SoftRock, such as *Rocky*, *KGKSDR* and *PowerSDR™*.

VK6VZ believes the degree of difficulty with building the RXTX v6.1 transceiver and getting it going is at least double that of doing the same with a SoftRock v6 receiver. Getting familiar with the SDR software is all-important and learning about the receive side of SDR without having to worry about transmitting is a great way to go.

Computing Power

G0BBL also makes the important point that whilst Softrock receivers can be used on a modest 500 to 700MHz Pentium personal computer running VE3NEA's popular *Rocky* software, as a minimum the RXTX transceiver needs a Pentium PC running at a speed of 1GHz to 2GHz, in order to be able to transmit.

So that the RXTX transceiver is able to transmit and receive, the soundcard of the personal computer that is doing the analogue-to-digital (and digital-to-analogue) conversion needs to have 'Line In' and 'Line Out' capabilities, as well as a microphone input socket.

Currently, there are three software packages that support the RXTX transmit capabilities. One is a special version of *PowerSDR™* – the software which is used on the Flex-Radio SDR-1000 - which has been adapted by Bill Tracy KD5TFD for this purpose. *KGKSDR* has been developed by Duncan Munroe M0KGK, while Rocky 3 has been developed by Alex Shovkoplyas VE3NEA – these are both transceive versions of software originally developed for the SoftRock series of receivers.

G0BBL created an excellent summary of the three SDR software programs for a talk he gave at the Ozarcon QRP conference in the USA, which is reproduced in **Table 8.8**, to help with choosing the software that is most appropriate to your needs. VK6VZ has tried them all and likes *Rocky* and *KGKSDR* the best – but then he is a simple soul who likes 'intermittent carrier' operation. VK6APH who lives on the cutting edge, but uses SSB, prefers *PowerSDR™*.

A certain amount of computer knowledge will be required when initially setting up the RXTX software program of your choice and the services of an experienced PC user who has an interest in SDR are bordering on essential. However, as G0BBL notes, once the RXTX transceiver and its software has been setup correctly, those radio amateurs with limited PC expertise should be able to operate the RXTX SDR transceiver.

Like the SoftRock receivers, there is a need with the RXTX transceiver to balance the phase and amplitude of the I and Q signals that come from the RXTX into the personal computer soundcard being used for analogue to digital conversion.

If you use the *Rocky* software, this balancing is automatically done on receive by the software itself, using signals that are received by the receiver outside of the bandwidth to which is tuned. Now when the RXTX is transmitting, I and Q signals are generated by the personal computer soundcard and the phase and amplitude of these transmit signals need to be similarly balanced.

For example, let us say the local oscillator crystal in the RXTX is on 1.830MHz and we wish to transmit on 1.845MHz. In this case, the I and Q signals from the soundcard will be at 15kHz, since:

1.830MHz + 0.015 MHz = 1.845MHz

In addition to the wanted signal on 1.845MHz, an image signal will also be present at the transmitter output at:

1.830MHz – 0.015MHz = 1.815MHz

Name of software and general description	Points in favour of its use	Points against its use
M0KGK SDR http://www.m0kgk.co.uk/sdr/index.php • Fully Featured Package by Duncan Munroe M0KGK. • RXTX transceiver supported since December 2006. • Current Version 1.59. • CW, SSB and PSK31 modes. • Separate Yahoo KGK support group.	• Excellent support by M0KGK. • New functionality – bug fixes implemented over weekend. • Supports single soundcard transceive operation. • Easy setup of WINPSK setup – VAC not required.	• High CPU usage – Modern PC needed (3 GHz upwards). • Filters are not optimum - CW signals can sound rough "off tune".
Rocky 3.1 http://www.dxatlas.com/rocky/ • By Alex Shovkoplyas VE3NEA. • RXTX transceiver supported since February 2007. • CW and PSK31 modes supported. • Built-in electronic paddle keyer. • Low latency. • Simple program – few controls - easy to get started with.	• Lowest CPU requirements (1 - 1.4 GHz Pentium PC). • Excellent filters. • Blocks transmission around carrier and outside +/- 90 % bit rate. • Manual calibration of transmit image rejection is easy and effective. • PSK 31 has built-in error detection and correction.	• SSB mode not supported. • Single soundcard transmit operation not supported. • Straight key not supported at this time.
PowerSDR™-sr40 http://sourceforge.net/projects /powersdr-sr40 • First RXTX version produced by Bill Tracy (October 2006). • CW and SSB modes supported. • Current version Soundforge 1.9.0. • Uses latest SDR1000 interface. • Contributed by Guido ten Dolle, PE1NNZ. • Extensive documentation.	• Fully featured – even has subreceiver support! • Excellent filters (try the 25Hz CW filter on a weak signal). • Modest CPU requirements. • Good documentation.	• Starting with *PowerSDR™* is bit of a learning curve.

Table 8.8: Comparison of SDR software. Note that this was written by G0BBL in early 2007 and should be considered as a general guide only - versions may have changed since then

Now, the amplitude of this image signal at 1.815MHz will depend on the accuracy to which the overall amplitude and phase of the I and Q signals have been matched. For example, if the amplitudes have been matched to 0.1dB and the phase to 1 degree, then the image will be 40dB below the wanted signal.

In practice, this means that if your signal - the wanted signal - is R5 S9+40dB at the receiver of a local station, the image of your signal will still be R5 S9 – a very strong signal.

Fortunately, as you can see by looking at Table 8.6, all the currently available software for the RXTX includes provision to reduce the unwanted image of the transmitted signal. One of the best ways of getting rid of the image – and feeling confident that it is gone – is by connecting the RXTX up to a dummy load antenna and listening for the unwanted image signal on another receiver.

Please bear in mind that this unwanted image adjustment needs to be done at a number of frequencies across the amateur band the RXTX is operating on – say, at every 5kHz - since the optimum settings will vary considerably across the band. Ultimately, a compromise setting may be necessary, for transmit I and Q phase and amplitude adjustment, that works across most of the band covered by your RXTX .

Each type of the three types of SDR software requires slightly different connections between the RXTX board and associated personal computer. However, diagrams showing the necessary connections for each piece of software are available on the Internet [58]. A few additional readily available components may be required.

On the Air

If you listen in the QRP CW and PSK sections of the 7MHz band, there is a good chance you will hear someone using a SoftRock RXTX transceiver. Routine CW contacts have been made over distances up to several thousand miles using dipole antennas. SSB contacts have also been reported and a number of operators are using the RXTX transceiver very successfully on PSK data mode.

Special thanks must go to Jan, G0BBL, John, G8BTR, Alan M0PUB, Tom, DG8SAQ and Bodo DJ9CS for their work in developing the RXTX, to Bill, KD5TFD, Guido, PE1NNZ, Duncan, M0KGK and Alex, VE3NEA for making software available for the RXTX, and to Tony, KB9YIG for his efforts in producing another great SDR kit at a very modest price.

USING THE SI570 DIGITALLY CONTROLLABLE CRYSTAL OSCILLATOR IN SDRS

There is something affordable in the world of SDR for just about everyone, rich or (relatively) poor. As explained in the previous section, the main-stay at the budget end of the SDR market has long been the terrific range of receiver (and more recently, low power transceiver) kits marketed by Tony Parks, KB9YIG [2].

This range of radios allows you to tune plus/minus half the sampling rate of your personal computer's soundcard from the crystal frequency selected by the user from the SDR hardware. In the case of the popular M-Audio D44 soundcard - used by both VK6APH and VK6VZ - this is ±48kHz. For example, if the crystal frequency is 7.040MHz, then using a 96kHz sampling rate the user can tune 48kHz either side of the crystal frequency, using SDR programs such as *PowerSDR*^{TM} [12], *Rocky* [1] and *KGKSDR* [3].

The main limitation of the early variants of SoftRock receiver lay in the relatively narrow bandwidth covered by the receiver, due to its fixed crystal oscillator. Whilst this was great for VK6VZ who only really uses the CW sections of bands, the majority

Fig 8.99: Silicon Labs Si570

(like VK6APH) who were interested in the wider open spaces of the SSB sections of the amateur bands and also liked to do a bit of general listening around the HF spectrum found this limitation rather frustrating.

As a result, the more experimentally-minded SoftRock users, in particular those members of the SoftRock 40 Yahoo reflector [13], started to search for a cheap and stable variable frequency source for their SDR receivers. One of the first used was the DDS-60 direct digital synthesizer kit [59] but although this is very reasonable priced, some were not keen on buying it as it was more expensive than the (very cheap) SoftRock receiver kits.

A very interesting alternative has appeared, as the result of experimentation by people such as Cecil K5NWA, Dave WB6DHW, Thomas DF7TV, and the QRP-2000 team (DG8SAQ, G8BTR, M0PUB, PE1NNZ, G0XAR and G0BBL).

The crystal oscillator in a SoftRock receiver or transceiver is replaced by a Silicon Labs Si570 programmable crystal oscillator integrated circuit (**Fig 8.99**). This clever chip can generate almost any frequency from 3.5MHz to 960MHz and selected frequencies up to 1,400MHz (although a minimum of 10MHz is guaranteed by Silicon Labs) and is programmable via an I2C serial interface in steps of less than 1Hz.

The Si570 can be controlled via any I2C device, either implemented in hardware or 'bit-banged' in software. This means a whole host of interfaces can be used - from PC printer and USB ports through to popular microprocessors such as the PIC range or even the tiny Atmel ATTiny45.

There are both CMOS (complimentary metal oxide semiconductor) and LVDS (low voltage differential signaling) output versions of the Si570. The CMOS version has a bigger output (2.6V peak-to peak) than the LVDS version (0.7V peak-to-peak), but the latter offers better stability (±20ppm) than the former ±50ppm) and lower phase noise.

A block diagram of the Si570 is shown in **Fig 8.100**. It consists of a PLL (phase-locked loop) whereby a microwave VCO (voltage

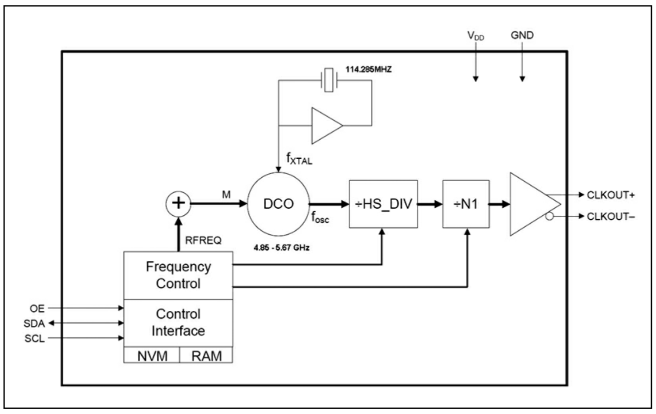


Fig 8.100: Block diagram of the Si570 controllable oscillator

Table 8.9: Si570 Output Phase Noise (dBc/Hz)

Offset Frequency	120MHz (LVDS)
100Hz	-112
1kHz	-122
10kHz	-132
100kHz	-137
1MHz	-144
10MHz	-150

controlled oscillator), covering the range 4.85 to 5.67 GHz, is divided down and phase-locked to an internal 114.285MHz (nominal) third overtone crystal reference.

Dividing the microwave VCO frequency results in a reduction in oscillator phase noise and typical performance figures are show in **Table 8.9**. Whilst the figures are not 'state of the art' they are very respectable for such a simple device.

A significant advantage of the Si570 is that unlike direct digital synthesis-based local oscillators, there are no spurs contained in the output spectrum. VK6APH has been testing a Si570 running at 1.126GHz as a local oscillator for a simple spectrum analyser with excellent results.

Whenever the frequency of the Si570 is changed, via an I2C command, the output is disabled for some 250µS. This results in 'blips' in the receive signal whenever the attached SDR is tuned.

However, Silicon Labs have recently updated the datasheet for the device, showing how frequency changes of up to ±3,500 ppm can be made without the need to disable the output. Experimental work by Sid, W7QJQ has increased this range to ±2.5 per cent of the set frequency, resulting in 'VFO-like' tuning of an SDR with a Si570 is attached to it.

Although the Si570 is stable enough for HF applications, Cecil, K5NWA, has come up a very simple and clever oven for it, attaching a PTC thermistor to the Si570's case in a similar manner to that used by VHF and microwave oscillators to stabilize conventional crystal oscillators. Details of this can be found on the web [60].

With a 60-degree thermistor such as a GE RL3006-50-60-25-PTO operating at 5V, the stability has been reported as being "as good as a ham transceiver with a temperature stabilized crystal oscillator (TCXO) option".

The QRP2000 team has developed a USB controlled crystal synthesizer using the Si570 for use with the SoftRock RXTX V6.1 and V6.2 transceivers, which is being sold by SDR-Kits.net [61]. Both CMOS and LVDS versions of this are available. The kit uses either a special version of *PowerSDR*-SR40 that has been developed by Guido, PE1NNZ, Alan, M0PUB, and others or a new version of VE3NEA's *RockySDR* (v3.5), which both provide USB support for the synthesizer. Tom DG8SAQ has written a companion stand-alone application for the SDR-Kits kit to set the frequency of the Si570.

Using the Si570 as the frequency source for SoftRock transceivers is a significant advantage over the original method because it alleviates the need to accurately balance I and Q signal components over a wide bandwidth.

As a result of the growing use of the Si570 and the SDR-Kits si-570 kit, Tony, KB9YIG developed a new multi-band SoftRock receiver - the vf8.3. This receiver included four bandpass filter modules for the band groups 3.5/7MHz, 10/14/18MHz, 21/24/28MHz, and 1.8MHz, but did include a Si570 module. Just as the writing of this chapter was being completed, Tony introduced a Softrock v9.0 Lite receiver kit with electronically-switched bandpass filtering that covered 1.8 - 30MHz. Contact Tony [2] for current details of SoftRock radios that can use the Si570.

REFERENCES

[1] The latest version of VE3NEA's *Rocky* software can be downloaded free from www.dxatlas.com/Rocky
[2] The SoftRock receivers and transceivers have (at the time of writing) reached Version 6. E-mail Tony Parks, KB9YIG, directly to check kit availability at: raparks@ctcisp.com. You can order from him by post or over the Internet/e-mail by using PayPal. Please note that Tony sells the kits for fun and any profits go to fund the development of new SoftRock kits.
[3] M0KGK's *KGKSDR* software can be downloaded free at: www.m0kgk.co.uk/sdr/index.php
[4] www.flex-radio.com
[5] Perseus - www.microtelecom.it/perseus/
[6] SDR-IQ - www.rfspace.com/RFSPACE/SDR-IQ.html
[7] ADAT ADT-200a - www.adat.ch/index_e.html
[8] Details of the High Performance Software Defined Radio Project (HPSDR) can be found at: http://openhpsdr.org
[9] QS1-R - www.philcovington.com/QuickSilver/
[10] The not-for-profit Tucson Amateur Packet Radio (TAPR) organization, www.tapr.org
[11] www.sherweng.com/table.html
[12] *PowerSDR*™ software & source code can be downloaded free at: http://flex-radio.com/Products.aspx?topic=PowerSDRv2
[13] You can join the SoftRock 40 reflector at: http://groups.yahoo.com/group/SoftRock40/ but you will need to become a Yahoo member first.
[14] You can join the Flex-Radio user reflector at: http://mail.flex-radio.biz/mailman/listinfo/flexradio_flex-radio.biz.
[15] Several I & Q .WAV files are supplied on the CD that comes free with this Handbook. Note that these are uncompressed I & Q files and are designed to be played using SDR software (eg *PowerSDR*™, *Rocky*, *KGKSDR*
[16] VK6APH used this technique in 'The Buccaneer - an experimental high-performance binaural receiver design', published in *RadCom*, July 2005.
[17] Flex-Radio sponsor regular Internet round-table discussion on the SDR-1000 and related topics using a VoIP (Voice over Internet Protocol) software package called 'Teamspeak' - for details see http://kc.flexradio.com/KnowledgebaseArticle50228.aspx
[18] QS1R web page - www.philcovington.com/QuickSilver/
[19] 'The Peter Hart Review: SoftRock v6', *RadCom*. March 2007
[20] 'SDR' columns, *RadCom*, May 2007, September - November 2007, January 2008
[21] Free software for SoftRock - see http://digilander.libero.it/i2phd/sdradio/index.html
[22] SDR Console software: www.philcovington.com/SDR.html
[23] SmallTalk project - see http://myweb.tiscali.co.uk/g3ukb/
[24] I2PHD - see www.weaksignals.com
[25] DttSP is an open source project started by Dr. Frank Brickle, AB2KT, and Dr Robert McGwier, N4HY, of the DTTS Microwave Society to provide code to be used in various DSP projects with an emphasis on Software Defined and Cognitive Radio - see http://dttsp.sourceforge.net/.
[26] www.nitehawk.com/sm5bsz/linuxdsp/linrad.htm
[27] http://javaguifordttsp.blogspot.com/
[28] www.ubuntu.com/
[29] www.apple.com/macosx/
[30] http://ewpereira.info/sdr-shell
[31] http://py2wm.qsl.br/SDR/SDRZero-2.html
[32] http://people.wallawalla.edu/~Rob.Frohne/SDR/SDR-Shell/

[33] www.nitehawk.com/w3sz/w3sz.htm
[34] www.nitehawk.com/w3sz/xlinrad-deb.htm
[35] http://jackaudio.org/
[36] http://groups.yahoo.com/group/dttsp-linux
[37] microembedded.googlecode.com/files/ATLAS_Docu_USLetterLowRes.pdf
[38] USRP boards - see www.ettus.com
[39] Details of the Xylo board can be found at: http://www.knjn.com/FPGA-FX2.html
[40] http://uwsdr.berlios.de/
[41] CDG2000 transceiver - see *RadCom*, June 2002 for part 1.
[42] There are several SoftRock application notes on using the SoftRock v6 for processing IF signals from analog radios. These can be found on the 'members only' part of the SoftRock reflector web pages, in the SoftRock v6 docs/v6
[43] Generally attributed to Gerald Youngblood, K5SDR, the original developer of the Flex Radio SDR-1000 (although used by others over many years).
[44] www.flex-radio.com
[45] Presentation by Kirk Weedman, KD7IRS, on Verilog programming: http://verilog.openhpsdr.org/
[46] Ozy circuit diagram - see at www.hamsdr.com/
[47] www.hamsdr.com/personaldirectory.aspx?id=485
[48] www.analog.com/library/analogDialogue/archives/30-3/single_chip.html
[49] 'Receiver Sensitivity, noise figure and dynamic range', James Fisk, W1DTY, *Ham Radio Magazine*, October 1975.
[50] 'IC7800 Test Drive', *QST* Magazine, August 2004, ARRL
[51] www.xs4all.nl/~martein/pa3ake/
[52] Open Office is an open source multi-platform, multi-lingual suite of office applications - see www.openoffice.org
[53] ScopeFIR: see www.iowegian.com/scopefir.htm
[54] An excellent explanation of the Fourier series can be found at: http://complextoreal.com
[55] FFTW - see www.fftw.org/
[56] Ooura FFT algorithm, see - www.kurims.kyoto-u.ac.jp/~ooura/
[57] www.mydarc.de/dg8saq/PAJan/index.shtml. IF applications folder. Analysis of RXTX 1W PA by Tom Baier, DG8SAQ
[58] A paddle and PTT connection diagram for Rocky 3 can be downloaded from the webpage at: http://www.dxatlas.com/Rocky/. A diagram showing the connections for KGKSDR and the RXTX can be downloaded from the RXTXv6.1 folder (see [2] above) - it is called Softrock RXTX connections.jpg. An excellent guide on how to setup *PowerSDR*TM *-sr40* with the RXTX by Guido PE1NNZ can be found at: http://sourceforge.net/projects/powersdr-sr40/files/powersdr-sr40/
[59] www.amqrp.org/kits/dds60/
[60] http://groups.yahoo.com/group/softrock40/files/Si570%20frequency%20stabilization/
[61] www.sdr-kits.net/

Audio, video and *Excel*TM files to complement this chapter can be found in the accompanying free CD

About the Authors

***Phil Harman**, VK6APH, has a BSc (Hons1) in Electrical and Electronic Engineering from the University of Bath (UK). He was first licensed as G3WXO at the age of 16 and became VK6APH in 1980 upon migrating to Australia. First trained as an RF Engineer, Phil has also held a number of IT management positions. For the past 10 years he has been developing advanced video processing technologies for stereoscopic imaging and the conversion of 2D images to 3D, and holds 23 patents relating to digital image processing. His main interests are receiver development, software defined radios, HEO satellites and HF DXing.*

***Steve Ireland**, VK6VZ, has been a scientific journalist for over 25 years and is a former editor of the UK's* Ham Radio Today *magazine. He was first licensed as G3ZZD at the age of 16 and migrated to Australia in 1989, where he became VK6VZ. He has a BA (Hons2) degree in English from the University of North London and also holds an Ordinary National Certificate in Engineering. His main interests are CW, low-band DXing (over 220 countries confirmed on 160 metres), antenna experimentation, software defined radio and writing technical articles about amateur radio.*