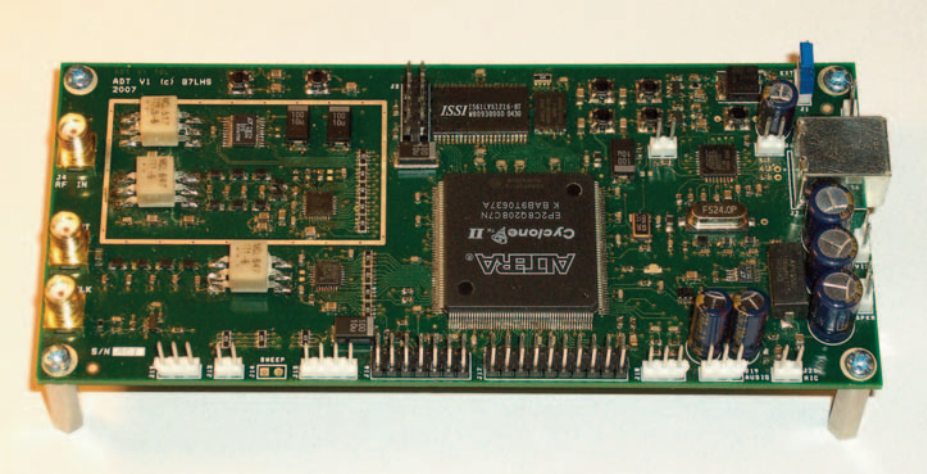
The All-Digital Transceiver

In this important article we look at an all-amateur, all-digital transceiver.



The All Digital Transceiver is a compact single board design.

MARCH OF PROGRESS. Integrated circuits are now available that can directly convert 0-30MHz analogue signals to and from the digital domain. This article describes a project undertaken by the authors to build the entire signal path of an HF transceiver digitally. The RF section is built on a 150x70mm printed card, with most of the functions carried out by a field programmable gate array (FPGA). The remaining signal processing is done in computer software. A Universal Serial Bus (USB) cable carries digital signals between the two parts and also powers the card. It can be used as a general-coverage HF receiver as it stands and the addition of a power amplifier and antenna switching will enable a complete transceiver to be constructed. The configuration of the FPGA logic is stored in a non-volatile memory on the card and can be upgraded from the computer.

This article describes the fundamental principles of the digital processes used, outlines the block diagrams of the receive and transmit paths and describes the inner working of the important blocks. The block diagram of the hardware is also presented, with details of the functions of the main integrated circuits. The modulation and demodulation processes that take place in the computer software may be described in a subsequent article.

FIRST PRINCIPLES. In the traditional superhet receiver, a mixer converts the incoming band to an intermediate frequency at which the channel filter selects the signal of interest. This is then passed to a demodulator to produce the final output. In mathematical terms, a mixer multiplies the signal by the local oscillator. If both signal and oscillator were pure sine waves and the mixer was a perfect multiplier, its output would consist only of the sum and difference signals or, to put it another way, the mixer would only respond to inputs on the wanted and image frequencies. However, real analogue mixers are imperfect and the best ones are more like switches than multipliers. Such mixers give excellent signal linearity but really bad 'oscillator linearity', which means they respond to signals either side of multiples of the oscillator frequency. For example, an analogue receiver with an oscillator on 1MHz and an IF of 100kHz will respond not only to signals on 900 and 1100kHz, but also to signals on 1900 and 2100kHz, 2900 and 3100kHz and so on. The magnitude of these additional responses depend on the purity of the local oscillator. All but one of the responses must be suppressed, usually with filters. The design of the complete receiver is thus constrained by the achievable filter performance. For example, the image rejection requirement puts a lower limit on the choice of the intermediate frequency.

Digital mixers, on the other hand, are perfect multipliers and do not exhibit oscillator harmonic responses. A digital front end using the above example frequencies would only respond to inputs on 900 and 1100kHz. There is a technique using two mixers, fed with 90-degree-shifted oscillators, in which the outputs of the two mixers are combined to cancel the image response. Although this technique can be used in the analogue world, the cancellation is far from perfect. However, a pair of digital multipliers, fed with digital sine and cosine waveforms, makes a perfect mixer with a single response. This removes the lower limit on the choice of intermediate frequency. Even an intermediate frequency of zero is possible.

This last statement may be a surprise. IF amplifiers at very low frequencies are not common in the analogue world because of problems with voltage drift and low-frequency noise, but these defects don't affect digital circuits. There is one practical difference between a conventional IF and the zero-frequency version: the two mixer outputs are not combined to form a single-response output, but they are kept separate and processed in parallel as two paths. This makes the demodulation process much easier and is sometimes known as the I/Q technique, where I and Q refer to Inphase and Quadrature, the names usually given to the two paths in such a process.

It's worth noting that in an I/Q path carrying a signal with a spectrum of width B, the signals in the I and Q paths each have a width of B/2. One could say that the spectrum of an I/Q signal extends either side of zero, in the same sense that the spectrum of a conventional signal extends either side of its centre. However, it's not true to say that the I channel represents one 'side' of the signal and the Q channel represents the other.

HARDWARE BLOCK DIAGRAM. At the top left of **Figure 1**, the receiver line-up starts with a transformer to match the 50Ω input impedance to 125Ω, the optimum figure for the low noise amplifier (LNA), an LT5514 device from Linear Technology. This is a 33dB linear amplifier preceded by a 16-step attenuator (1.5dB/step), which is controlled by four lines driven from the FPGA.

Another transformer couples the LNA to the balanced 30MHz low-pass filter, which is a 9th order Chebyshev design that aims to keep the alias response of the ADC below -100dB.

The ADC, an LTC2254 device from Linear Technology, has 14 data bits and an extra 'overload' bit. Fourteen bits can only handle 84dB, which might not seem enough to capture the weaker signals, but in fact the total input to the ADC (in a 30MHz bandwidth) is always much larger that the smallest digitisation step. The 14bit limitation merely shows up in the digital data as noise

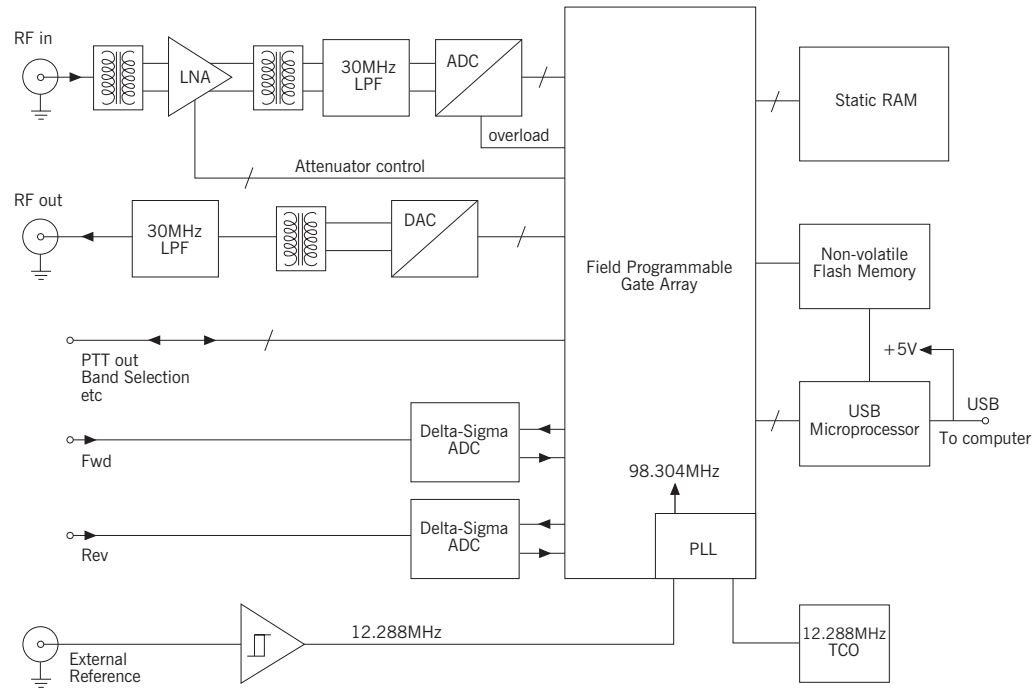


FIGURE 1: All Digital Transceiver hardware block diagram.

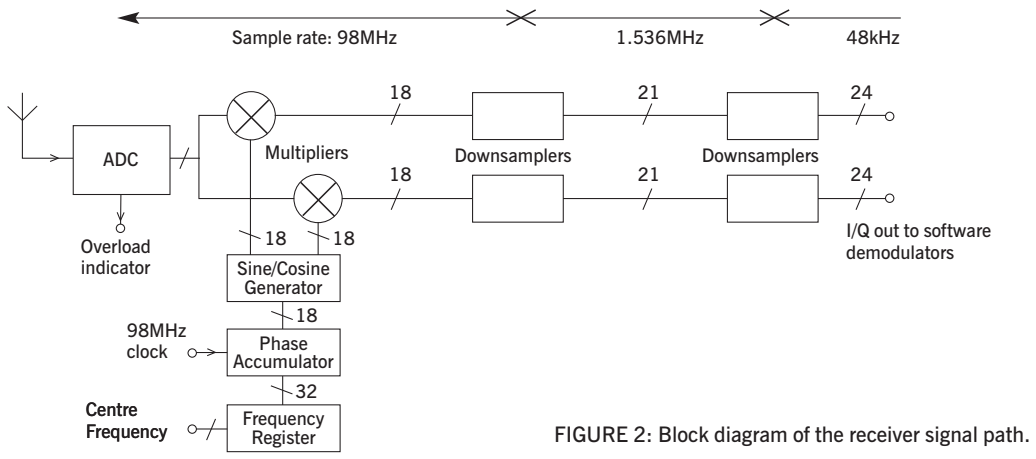


FIGURE 2: Block diagram of the receiver signal path.

spread uniformly across the whole 30MHz band. Noise 84dB down in a bandwidth of 30MHz equates to noise 124dB down in a bandwidth of 3kHz. This means we can copy narrow-band signals way below the smallest digitisation step. 14 bits is plenty.

On the transmit side, a 14-bit 210MHz DAC, an AD9744 from Analog Devices is followed by a step-down transformer and a single-ended 9th order Chebyshev filter, to deliver up to 10mW output into 50Ω.

The FPGA, an Altera Cyclone II type EP2C8, contains 8256 general-purpose logic cells and 18 multiplier cells, as well as about 160k bits of memory. Some 1600 of these cells make up a 32-bit microprocessor which handles many of the housekeeping functions. A separate static RAM chip stores program and data for this processor. Two phase-locked loops generate the internal clock signals, driven from either an on-board TCXO or an external reference. The logic configuration is loaded from a non-volatile FLASH memory at power-up. In addition to interfaces for the ADC and the DAC, there are a number of general-purpose I/O pins for bandswitching, PTT and other things, some of that we haven't thought of yet. The forward and reverse power sensors use an input and an output pin, together with a few resistors and capacitors, to implement a delta-sigm a converter.

The USB interface is handled by an Atmel AT89C5131A, which is a 24MHz microprocessor with on-chip non-volatile program storage. An 8-bit bi-directional bus communicates with the FPGA. This device can be programmed over the USB from a computer and also interfaces to the FLASH memory so that the FPGA configuration can also be uploaded from a computer.

The printed circuit is a four-layer construction with one continuous ground plane.

THE DIGITAL RECEIVE PATH. In **Figure 2**, the analogue to digital converter (clocked at 98.304MHz) is preceded by a 30MHz low-pass-filter (not shown). This ensures that the receive will hear everything up to 30MHz but not respond to anything in the range 68.304 – 98.304MHz, which would 'alias' down to the wanted signals below 30MHz and cause QRM. The digital output of the ADC is taken to two multipliers in the FPGA, the other inputs of which come respectively from a digital sine wave and a digital cosine wave.

THE SINE-COSINE OSCILLATOR BLOCK.

This is probably the most important block in the receiver **[1]** because any defects in the oscillator signals will give rise to unwanted responses in the complete receiver. This block works very like a direct digital synthesiser (DDS), but without the digital-to-analogue conversion. On each cycle of the 98.304MHz clock, a 9-bit frequency register (FR) is added to a 9-bit phase accumulator (PA), which addresses a 512-step lookup table containing a full-cycle sinewave. To generate a frequency of 192kHz (98.304/512MHz), for example, the FR is set to 000000001 and the lookup table pointer will thus step by one position every cycle and trace out a 512-step sine waveform at 192kHz. The matching cosine waveform is formed by another pointer that is 90° 'further round' the table.

The FR is set to other values to generate other multiples of 192kHz. However, a tuning step size of 192kHz is far too coarse to be useful, so the FR and the PA are extended by another 9 bits, to 18 bits. However, only the top 9 bits of the PA are used to address the lookup table. The effect is a tuning step size of 375Hz. To generate a frequency of 192.375 kHz, a '1' is set in the 9th bit and a '1' in the 18th bit of the FR. This results in an extra step in the lookup table pointer every 512 cycles. The resulting output has a frequency of 192.375kHz, but the repeating extra step shows up as a pair of spurious sidebands 375Hz either side, at about 54dB down.

These sidebands are reduced further by using the additional 9 bits in the PA to interpolate smoothly between the lookup table steps. Calculus theory shows that the

difference between two adjacent sine values is proportional to the corresponding cosine value and this makes it easy to calculate the required adjustment. The cosine value, as read from the lookup table, is first multiplied by a constant and then by the interpolation fraction, that is, by the second set of 9 PA bits. This is added to the looked-up sine value to give the interpolated value. The interpolated cosine is done in a similar way. To economise on the number of multiplier blocks needed (there are a limited number of these in the FPGA), a second lookup table is used to store all 512 multiples of the constant. This interpolation technique takes the worst-case spurii down to -108dB. This figure sets the ultimate unwanted response level of the entire receiver.

The tuning steps can be made much smaller than 375Hz by further extending the FR and the PA (eg to 32 bits). This introduces more spurii, but none are worse than -108dB.

THE DOWNSAMPLER BLOCK. The outputs of the two multipliers go to the two-path 'zero-IF strip' at a sample rate of 98.304MHz, but this rate is far too high for comfort. The bandwidth of the signals of interest are only a few kHz and the bandwidth needed in the I and Q channels is half that of the signal itself, so the sample rate required by the demodulation software is itself only a few kHz. For this project we chose to feed the data down the USB cable to the computer at 48kHz. This is enough to handle broadcast quality signals and even a modest bandscope. The reason for the odd choice of 98.304MHz for the RF sample rate now becomes apparent – to get from here to 48kHz means reduction by a factor of 2048, which is a 'nice' number in digital terms.

Nyquist says that before the sample rate can be reduced to 48kHz, all traces of the input spectrum above half this frequency must be removed. If this is not done, unwanted signals around multiples of 48kHz will be 'aliased' down around zero and will interfere with the wanted signal. The downsampler must therefore incorporate a lowpass filter.

The simplest form of digital lowpass filter forms the average of a run of N input samples. If N is chosen to be 2048, the frequency response looks like **Figure 3**. It's not a brilliant filter but note that the notches between the sidelobes are at precise multiples of 48kHz and this is exactly what is needed to reject the alias products that also occur at multiples of 48kHz. The notches are narrow, however, so although an unwanted signal exactly 48kHz away would be perfectly notched-out and wouldn't interfere with a wanted signal on the centre frequency, an unwanted signal 47 or 49kHz away would not be completely rejected. Care must therefore be taken to ensure that this doesn't become a problem at the edges of the wanted passband.

The running-average process is very easy to do in an FPGA. Input samples are summed continuously in a digital integrator. At intervals of 2048 samples a copy of the integrator output is saved. The difference between consecutive saved copies is the desired output. If one filter doesn't give enough alias rejection at the passband edges, two or more can be cascaded. This is also easy since it is possible to combine the integrators into one block running at the input rate and the differentiators into another block running at the output rate; the end result is the same. This technique is known as a Cascaded Integrator Comb (CIC) filter **[2]** (the differentiators behave like comb filters). Another technique, which helps to minimise the number of logic gates needed to achieve a given performance, involves reducing the sample rate in several stages. For this project we chose to downsample from 98.304MHz to 1.536MHz with 3 CICs, then down to 48kHz with 7 CICs. **Figure 4** shows the passband shape and the first alias response (the worst one), plotted against frequency. This graph was generated with a computer simulation of the filter itself (the 'noise' at about -144dB is the resolution of the 24-bit arithmetic). We decided that ±5kHz was the widest passband we would need in the complete receiver, so we are not concerned about the alias response levels further out and -130dB at 5kHz is acceptable. The wanted response ends up with a slight droop, about 1dB at 5kHz, but that can be dealt with in the computer.

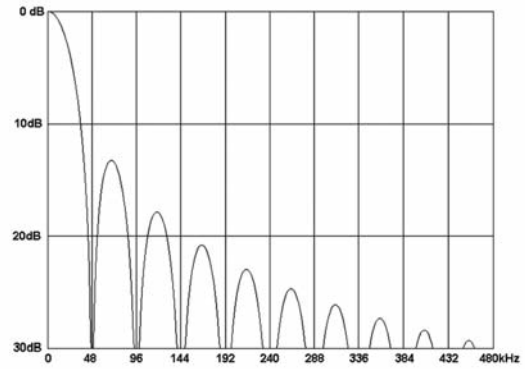


FIGURE 3: Frequency response of a run-of-2048 filter clocked at 98.304 MHz.

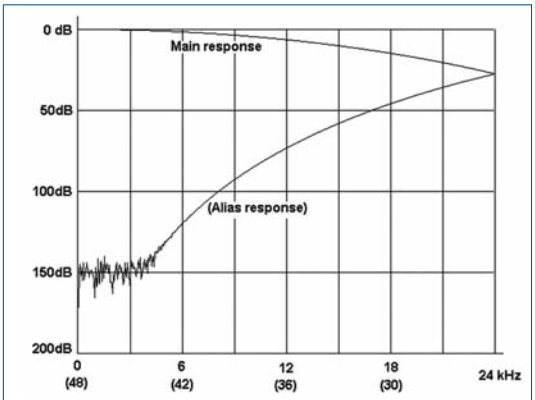


FIGURE 4: Overall response of the receive downsample filter showing the first alias response folded back to 48kHz.

The two 48kHz output streams are fed down the USB cable to the computer, where the software defines the finished passband precisely and carries out the desired demodulation.

THE DIGITAL TRANSMIT PATH. The transmit side (**Figure 5**) does not take so long to describe. It is most easily explained backwards since the process mirrors that of the receiver. A digital to analogue converter (DAC) generates the RF output at a few mW and this goes direct to the output connector via a 30MHz lowpass filter. The input to the DAC is derived from the sum of two multipliers exactly like those on the receive side, each one multiplying one IF signal by one of the outputs of a sine/cosine block exactly like the one in the receiver. The discussion on receiver spurious responses applies equally to the transmitter – just replace the word 'response' by 'emission' wherever it appears.

The need to downsample the receive path is mirrored by the need to upsample the transmit signal. Upsampling is done with exactly the same type of CIC filter that was used in the receiver, but with the integrate and difference processes transposed. Connected this way round, these devices can be thought of as interpolators rather than filters. Once again, the discussion on the alias responses in the receive downsampler also applies to the transmit upsampler in respect of spurious emissions. The slight droop in the passband is also present in the transmitter, but rather than pre-compensate for this in the computer software, we chose to do it in the FPGA with a 3-tap delay-line (FIR) filter.

There is another difference between the receive and transmit paths in our design. The device we chose for the USB interface can only handle one 48kHz I/Q stream so we dropped the transmit sample rate down to 16kHz. This means that the line-up of the transmit upsampler is not quite the same as that of the receive downsampler.

To complete the digital transmit function, we have placed a gain-control element (another multiplier) between the output mixers and the DAC. This performs the essential transmitter ALC function. We chose to put it here rather than in the computer software so that the drive to the PA can be reduced quickly in the event of an overload, regardless of how slow is the loop back via

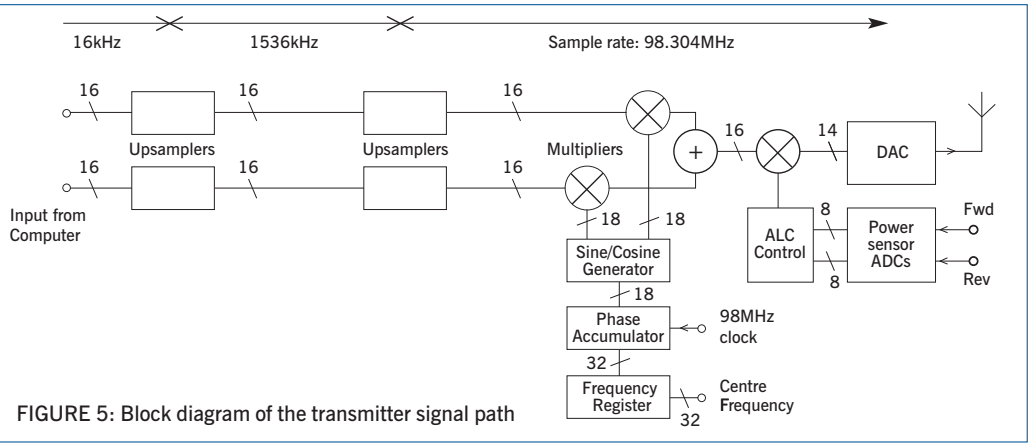


FIGURE 5: Block diagram of the transmitter signal path

the computer. To complete the ALC function, there are two auxiliary low-spec ADCs on the card, implemented in the FPGA. These take DC voltages in the range 0-3.3V derived from conventional forward and reverse power sensors in the transmitter PA.

Note that the transmitter and receiver can operate simultaneously. This leaves open the interesting possibility to develop techniques for improving the transmitter PA linearity.

FIRMWARE. The FPGA logic was written in the Verilog language using the Quartus II design software. The code for the USB processor was written in the C language and that for the 32-bit processor in C++. All this represents about 800 hours work, carried out by Steve, by far the largest part of the total workload in the project so far.

PERFORMANCE MEASUREMENTS. At the time of writing only the receiver has been measured. We should be should be to publish the transmitter measurements at a later date.

All measurements (except noise figure) were made at 14MHz with the LNA attenuator at zero.

Receiver sensitivity (3kHz bandwidth)
-116dB (0.35μV) for 10dB S/N

Noise figure
1.8MHz 13dB
14.1MHz 12.9dB
29.5MHz: 17.2dB

Phase noise
@5kHz -127dBc/Hz
@50kHz -132dBc/Hz
@500kHz-127dBc/Hz

The rise in phase noise at 500kHz is unusual, but not in itself a poor figure.

Third-order intercept point, usually quoted for analogue receivers, is of dubious value in the context of digital receivers, since the unwanted products in a digital receiver do not obey a third-order law [3]. However, everyone asks us what it is, so we had to do the measurement. It was +31dBm at all tone spacings and is almost certainly set by the LNA. The second-order intercept point measured +60dBm.

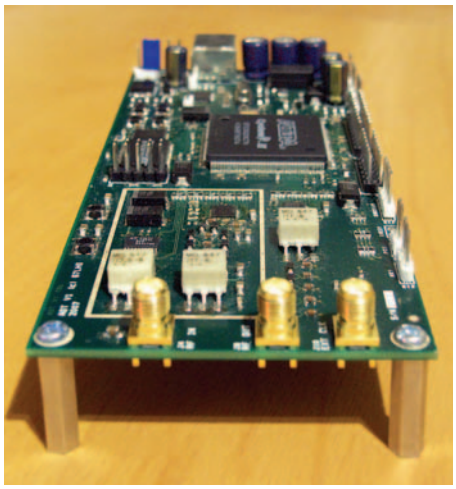
With the LNA at maximum gain the ADC reaches full-scale with a single-tone input of -20dBm.

FINISHING THE JOB. To make a complete transmitter/receiver using the All-Digital Transceiver (ADT) card as developed so far, a computer with a USB interface is required. As well as the receive and transmit data paths, there are some auxiliary control functions that share the USB, for example the programming of the FPGA is done this way as mentioned earlier. Setting the frequency registers in the sine/cosine oscillators is done via the USB, as is the control of the receiver input gain/attenuation. Up to now we have used the receiver without sub-octave bandpass filters and not encountered any overload problems, but there are spare pins on the board to switch such filters. They will be needed in any case for switching transmitter low-pass filters. In the other direction, we can foresee that it will be necessary to signal such things as the receiver ADC over-drive indication and the ALC sensor readings.

In a subsequent article, we plan to describe how the computer software processes the receive data, first to define the channel width to match the signal being received and then to demodulate it. The SSB demodulation process will be described of course, but other modes will be covered for completeness, since two-path zero-IF equivalents of traditional demodulation techniques will be new to many readers. We will show, for example, how to perform the apparently impossible task of demodulating FM from a carrier whose centre frequency is below the audio band! The matching transmitter modulation functions will also be described, together with some of the other features normally found in modern transceivers, such as noise blankers, notch filters and speech processors. This article may also describe further developments to the ADT board hardware and firmware.

The flow of receive and transmit data between the ADT board and the computer is very similar to that of audio data between a computer and a semi-professional soundcard, but we have run into problems using the soundcard application interface in the Windows operating system. Nevertheless, we have the transceiver operating on the air at the time of writing (September 2008). By the time the second article appears we should have overcome these problems, so it should then be possible to drive the ADT board using Software Defined Radio programs of the kind that are already available for SDR designs using analogue RF and soundcards. We don't believe that the ADT board is suitable for home construction by the average amateur, but, in due course, we hope to be able to give details of the availability of assembled boards, should there be a demand for them.

ACKNOWLEDGEMENTS. We are grateful to Colin Horrabin, G3SBI, Dave Roberts, G8KBB and George Fare, G3OGQ, who did the performance measurements, to Anne Carrington at Linear Technology for help with the LT5514 and to James Ross and David Evans of EBV for help with the FPGA.



SMA connectors are used for the RF In, RF Out and external clock. Prototype boards were hand-soldered, a difficult task for the average home constructor.

REFERENCES

[1] The sin/cos idea came from work by Pawel Jalocha, SP9VRC.
[2] Hogenauer, E. "An economical class of digital filters for decimation and interpolation," IEEE Trans. Acoust. Speech and Signal Proc., Vol. 29 No 2, pp. 155-162, April 1981.
[3] Leif Asbrink, SM5BSZ. "IMD in Digital Receivers," QEX Nov/Dec 2006, pp 18 – 22.

WEBSEARCH

www.sm5bsz.com/dynrange/qex/digital-imd.pdf

All Digital Transceiver part 2

The series continues with a look at the software and mathematics of receiving

RECAP. Part 1 of this article appeared in the March 2009 *RadCom*, and described the RF unit of the authors' all-digital transceiver project. This described how the high-speed digital circuitry in the RF unit 'tunes in' a band of frequencies around a chosen centre frequency in the 0-30MHz range, and passes this in a digital form down the USB cable to the computer. This part will describe how the computer software processes this digital data to the point where it can be heard in the computer speaker. Part 3 will describe the path from the microphone (or from audio-based digital-mode software) through the computer software and out to the USB interface to the RF unit. Part 3 will also describe some of the techniques for interfacing the transmit and receive audio signals digitally, for example to/from other digital audio processing software.

POSTSCRIPT TO PART 1. In March we said we hoped to give details of the availability of assembled boards. We cannot do this yet. It's possible that the board may need to be redesigned to accommodate a later version of the FPGA chip, and this would also enable us to upgrade to a faster, more powerful USB chip, which will in turn make it possible to incorporate the microphone and speaker interfaces on the ADT board. More about this in part 3.

THE COMPUTER SOFTWARE. The digital data from the RF unit to the computer can be treated as if it was equivalent to the intermediate frequency signal of a conventional analogue superhet receiver, but with the IF frequency being centred on zero rather than at a frequency well above the audio band. We hinted that the DSP software in the computer would be able to handle this strange 'zero IF' concept. The key feature that makes this possible is that the digital data from the RF unit to the computer takes the form of two independent channels, known as the Inphase (I) and Quadrature (Q) channels. In the RF unit the I channel is derived by multiplying the RF input signal by the cosine output of the oscillator block and the Q channel by multiplication with the corresponding sine output.

The receive I/Q data from the RF unit is transferred to the host computer via the USB connection formatted as a pair of 24-bit signed binary numbers, transmitted at a rate of 48,000 pairs per second. This is the same format that is used for stereo audio. This makes it fairly easy to write the computer software to use the same operating system interface that would be used by software intended for stereo sound processing.

Nyquist tells us that a signal sampled at 48kHz can convey information in a band from 0 to 24kHz. The usefulness of the I/Q format is that we can treat the combination of the two channels as if it represented the band of signals extending up to 24kHz on both sides of the 'zero' centre frequency, mirroring the way that signals at the antenna are 'both sides' of the HF centre frequency in the RF unit. In this part we will describe how we process and demodulate signals in this form.

A BAND-SCOPE. Before we think about demodulating the signal, we can digress to process the 48kHz-rate I/Q data from the RF unit as a spectrum or waterfall to display on the screen. This is done by feeding the I and Q data into two buffers. At suitable intervals (determined by the rate at which we want to update the display), the last 1024 I and Q values in these buffers are processed by a Fast Fourier Transform (FFT) algorithm. This is a mathematical process that transforms the input time-domain data into the equivalent frequency-domain form. The clever thing about using an FFT with I/Q data is that the spectrum output displays the band both sides of zero, so the band-scope display is already centred on the dial frequency. A 1024-point FFT at 48kHz is capable of a resolution of about 50Hz.

Although Nyquist says that we should be able to make a bandscope that extends to ±24kHz, the RF unit filtering has narrowed the useful bandwidth, leaving us with a response some 30dB down at ±24kHz. We chose to display ±9kHz, at which bandwidth the RF unit response is a few dB down. This droop can be corrected in the display itself. Note that the dynamic range of this bandscope is rather more than we would get with a spectrum display on an analogue receiver – the vertical extent of this display is a full 140dB.

THE MAIN RECEIVER PASSBAND. For HF working we are usually interested in signals with bandwidths of a few kHz. The signal from the RF unit is rather wider than this, so before we start to think about demodulating it we must pass it through a filter of the chosen width, for example 2.7kHz for SSB or 6kHz for AM. The simplest way to do this is to pass the I and Q signals through identical low-pass filters, the cut-off frequency of which is chosen as half the required bandwidth, for example 1.35kHz for the SSB filter. This is done with the same basic 'summing of delays' structure that was used by the Cascaded Integrator Comb (CIC) filters in the RF unit. There we were primarily concerned with reducing the sample rate and suppressing the attendant

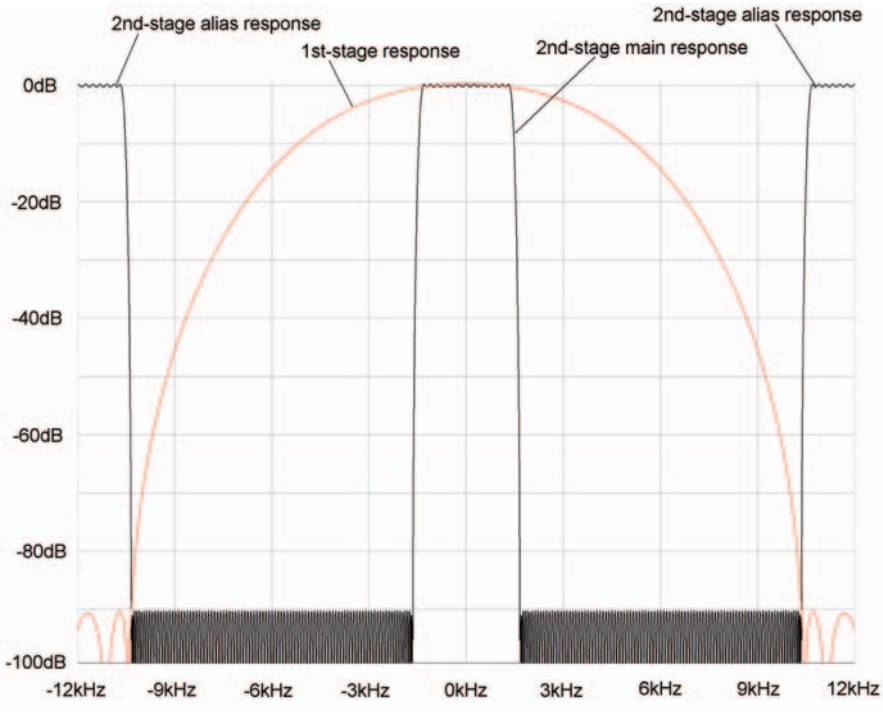


FIGURE 1: Two-stage SSB filter.

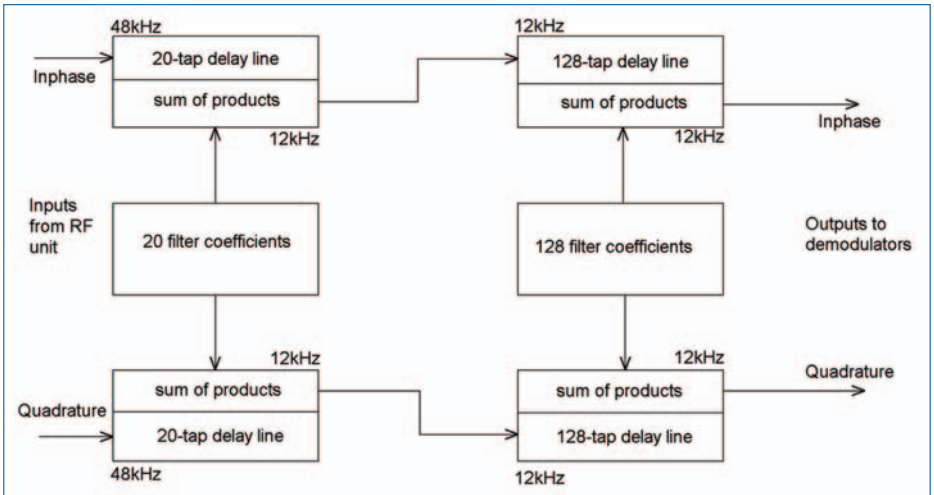


FIGURE 2: Channel filter flowchart.

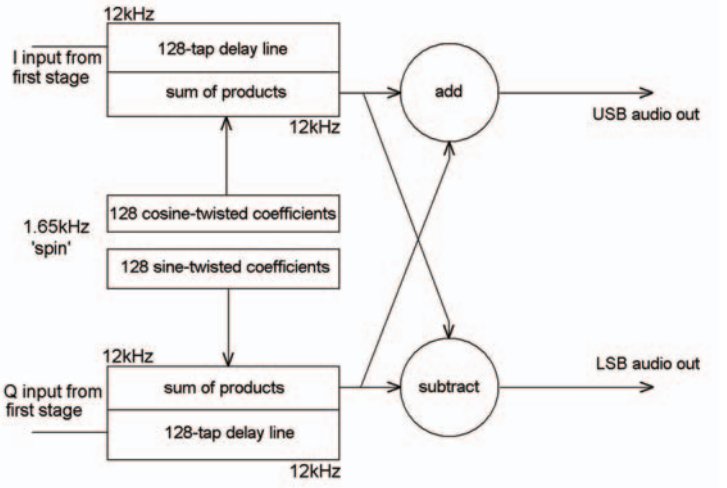


FIGURE 3: The SSB filter and demodulator.

alias responses, and not too worried about the shape of the main response. Here, though, we really do want a flat top and steep sides, so we need a more complex set of filter coefficients – not just the summing process that we used in the RF unit CIC filters.

A typical SSB filter passes signals up to ±1.35kHz and rejects everything outside ±1.65kHz. To do this with a tapped delay line at a sample rate of 48kHz we would need something like 450 coefficients, which would require a total of 43 million multiplies per second (450 x 2 x 48000). Since the signal from the filter is just a few kHz wide and the Nyquist limit is just over twice this value, we can reduce the computation requirement considerably by doing the filter in two stages. The second stage (and the demodulator that follows it) can be at a lower sample rate. 12kHz was chosen for the second stage. At this rate only 128 coefficients are needed to make a good SSB filter, and that corresponds to just 128 x 2 x 12000 = 3 million multiplies/sec.

The first stage filter, at a sample rate of 48kHz, is needed to reject signals around ±12kHz before we can reduce the sample rate to 12kHz. Since we want the finished filter to reject everything outside ±1.65kHz, the lowest 'alias' response that we must suppress is at 12-1.65kHz = 10.35kHz. The first stage filter must also be flat up to 1.35kHz, but it only needs about 20 coefficients to achieve this performance, which comes to 1.92 million multiplies/sec.

We can economise even further when we notice that we only need to calculate the output of the first-stage filter at the second-stage rate. This brings the total number of multiplies down to 3.5 million/sec, a useful improvement over the single-stage design. The frequency responses of the two stages, in bandpass form, are shown in **Figure 1** and the flow diagram in **Figure 2**. The filters in Figure 1 were designed using the Parks-McClellan method, which is widely available **[1]**. This method delivers a filter with equal amplitude passband and stopband ripples.

In an analogue receiver, the channel filter would normally be followed by an IF amplifier incorporating a gain control of some kind, not just for listener comfort but to avoid overloading the later stages. With DSP we can easily handle the full dynamic range (up to 384dB with floating-point arithmetic!) right up to the speaker output, so we don't need to put the AGC control just downstream of the channel filter if we don't want to. We will do it later.

SSB DEMODULATION. At this point our signal is still a pair of I and Q signals that extend from zero to 1.35kHz. A received SSB signal would be centred on zero with its carrier offset by 1.65kHz. We clearly have more work to do before we can listen to the demodulated audio.

In a previous article **[2]**, a DSP-based phasing-type SSB modulator was described, using a pair of 'twisted' bandpass filters to implement the 90° phasing network. The same technique can be used for SSB demodulation of the I/Q signal in this receiver. In the phasing demodulator the outputs of the two bandpass channels are added together for USB reception and subtracted for LSB. But we have already designed the main filter and we don't want to introduce further filtering just for the purpose of generating the 90° phase shift, and there is still the 1.65kHz carrier offset to deal with.

The solution is to transform the existing filter into a 'twisted' form. The outputs of the two twisted channel filters can then be added and subtracted to give USB and LSB outputs in the same way as is done in the phasing method. The chosen transform also has the useful property that it can slide the passband off-centre by any chosen amount, for example by 1.65kHz. The finished demodulator can now receive USB or LSB (or both) with a central suppressed carrier frequency.

Without going into the mathematics too deeply, the twist transform involves multiplying the set of I-channel filter coefficients by a sequence of cosine values and multiplying the set of Q-coefficients by the corresponding sine values. The sine and cosine sequences 'spin' at the chosen offset frequency.

Because of the twists applied to the filters, the coefficients are no longer the same for both channels, so Figure 2 isn't quite right now. **Figure 3** shows the modified arrangement, including the add and subtract operations used to form the USB and LSB outputs. Note that the twist calculations need only be done once when the filter is designed, although there is an interesting possibility for recalculating the offset frequency while receiving and thus implement 'passband tuning'.

Having demodulated the SSB signal, we are about to feed it to the speaker but we must first implement the AGC that we postponed earlier. This is much easier in DSP than in an analogue design - we just divide the signal by its peak value. We can also use this value for the S-meter display, after converting it to decibels or S-points.

AM/FM DEMODULATION. The receive path for AM and FM starts with the filter arrangement shown in Figure 2, except that the 2nd-stage filter is 6kHz wide rather than 2.7kHz. In part 1 we hinted that it is possible to demodulate these modes from a zero-IF signal without the carrier appearing in the output. To understand how this is done, it helps to visualise the I and Q components of a signal as the rectangular co-ordinates of a point in a plane. **Figure 4** shows a graph with the axes labelled as I and Q. Any signal can be represented as a dot somewhere on this graph. The origin represents the 'no-signal' condition and a stationary dot

at (1, 0) represents an I signal value of +1 and a Q value of 0. If we work back to see what signal at the antenna would give this combination of I and Q, it's a carrier on the receiver centre frequency with an amplitude of 1 at zero phase. If this carrier is modulated in amplitude to a depth of 100%, the dot in the I/Q plane will swing along the I axis, outwards to the point (2, 0) and inwards to the origin. If we want to listen to the modulation, we could, in this particular case, just feed the I channel to the speaker after removing the DC component.

However, if the carrier phase changes by 90°, the dot moves to the point (0, 1). The I channel signal is now zero and it's the Q channel that contains the modulation. If the unmodulated carrier moves 100Hz low of the centre, it becomes a dot spinning anticlockwise around the black circle in Figure 4 at 100 revolutions/sec and such a carrier modulated with a 1kHz sine wave traces out the red squiggle, which shows 9 cycles of modulation as the carrier spins. How do we recover the AM modulation in all these cases, and at the same time make sure we don't hear an off-frequency carrier as a tone in the speaker?

We calculate the length of the hypotenuse of the right-angled triangle formed by the I and Q values. For a signal represented by the point P in Figure 4, the I value is the distance OA, the Q value is the distance AP, and the value we want is the distance OP, which Pythagoras tells us is the square-root of the sum of the squares of the I and Q signals. It is this value that we pass to the output. Not only do we get the correct output regardless of the carrier phase, but since an unmodulated off-frequency carrier traces out a perfect circle around the origin, the output signal contains no trace of the 100Hz 'beat-note' that is present in both the I and Q channels individually. We can tune right through the passband and not hear a heterodyne.

The hypotenuse signal contains a DC component that must be removed before we pass it to the speaker, but we don't discard it completely. This value is proportional to the carrier level, and if we divide the recovered audio by it, the output is independent of signal strength. AGC is really easy with DSP!

The I/Q plane representation also helps to us see how we demodulate FM. Instead of finding the length of the hypotenuse, we calculate the angle between it and the I axis. Ignoring the AM modulation shown in Figure 4, this is the angle POA. The tangent of this angle is the ratio of the distance AP to the distance OA, and it is the inverse tangent function that we use to calculate the phase angle. This 'arctan' function only works properly over a 180°arc, but there is a special 'arctan2' function that takes the I and Q values individually rather than as a ratio, and this can be used over the full circle. This gives us the signal phase. Frequency deviation is the rate of change of phase and we can derive this by subtracting the phase calculated at each signal sample from the phase of the previous sample. Since an unmodulated off-frequency carrier will be rotating at a constant rate, it will just appear as DC at the output, so once again there is no sign of a heterodyne as we tune through the passband. The FM demodulator output is already independent of signal level so no AGC is needed.

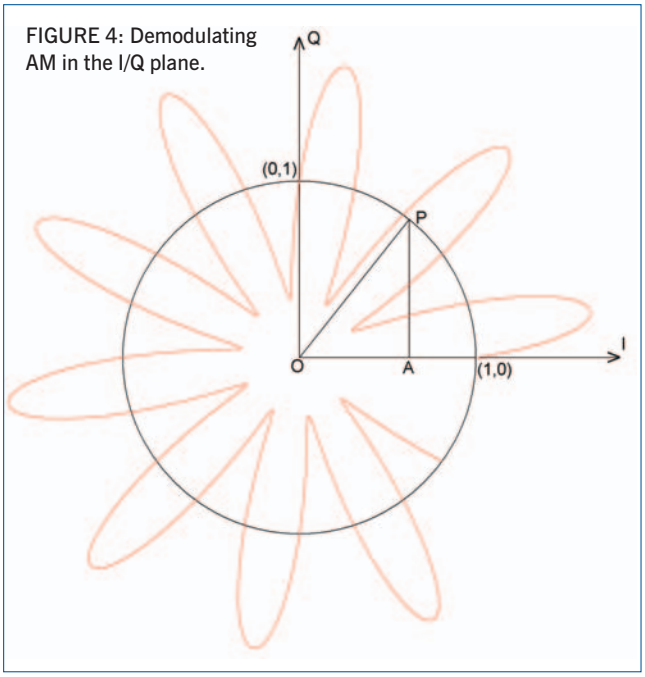


FIGURE 4: Demodulating AM in the I/Q plane.

CW RECEPTION. If we go back to Figure 2 and use a pair of 50Hz lowpass filters in the 2nd stage, we get a 100Hz wide passband centred on the 'dial' frequency. To make this sub-audio I/Q signal audible in the speaker, we transpose it up into the audio band by forming the quantity I*cos(2*ã*F*t)+Q*sin(2*ã*F*t), where F is the ratio of the desired 'CW pitch' to the sample rate and t is a number that increases by one for each sample processed. Regardless of the chosen CW pitch, the passband stays on the dial frequency. AGC is done in the same way as for SSB.

We have now demonstrated demodulation of all the traditional modes from zero-IF signals. Even if someone invents a new modulation method that we don't know about yet, we can be confident that we will be able to write the DSP software to demodulate it.

A NOISE BLANKER. High-amplitude, short-duration RF spikes can often be encountered, especially in urban environments. In the traditional analogue receiver, a broadband detector in an early part of the signal path can be used to mute the later narrow-band section, and this helps to reduce the disturbance caused by these spikes. The output from the RF unit is about 20kHz wide, which is enough to let us do the noise blanker in software.

A problem occurs with noise blankers of this type if there is a strong adjacent signal within the blanker bandwidth. The fast edges of the blanker gate, modulating the strong adjacent signal, cause it to 'splatter' across the wanted signal. In this situation a conventional noise blanker can often make the situation worse rather than better.

To overcome this problem we can slow down the blanking action, so that the rise and fall times are rather longer than those of the spikes we are trying to suppress. This will reduce the splatter sidebands around the strong adjacent signal. However, for this to be effective we must start the slow blanking action before the arrival of the spike that we are trying to suppress. We cannot predict when the spike will arrive, so we can only attempt this technique if we also delay the signal path.

In the analogue world this requires the use of an expensive broadband delay line but it's very easy to make delay lines in DSP. The incoming 48kHz I/Q signal goes through a 1ms (48-sample) delay line, and we examine the signal half way along. If we detect that there is a high-amplitude short-duration spike here, we apply a smoothly-curved blanking function to the entire delay-line, which completely blanks the middle samples (containing the spike) and leaves the two end values at their original levels. The 1ms delayed output thus has the spike removed but surrounded by a rounded gap which doesn't cause adjacent channel signals to splatter.

This completes the descriptions of some of the DSP techniques that have been used in the receive side of the software section of the All-Digital Transceiver. In part 3, the transmitter processes will be described, working through from the microphone audio input through the modulators, ending with the I/Q data stream that is fed along the USB cable to the RF unit for upconversion to the desired RF frequency. Part 3 will also go into the details of some of the processes needed in the digital transceiver for which there are no equivalents in the analogue world, specifically the problems of synchronising digital audio inputs and outputs and the techniques needed to interface between the transceiver and other audio-based software such as might be used for modes like RTTY or slow scan TV.

REFERENCES

[1] www.dsptutor.freeuk.com/remez/RemezFIRFilterDesign.html or just Google for "Parks McClellan"

[2] A digital SSB phasing network. Peter Martinez G3PLX. *RadCom* June 2004 p84.

All Digital Transceiver part 3

In this third and final part we look at the mechanics of transmission and timing issues.

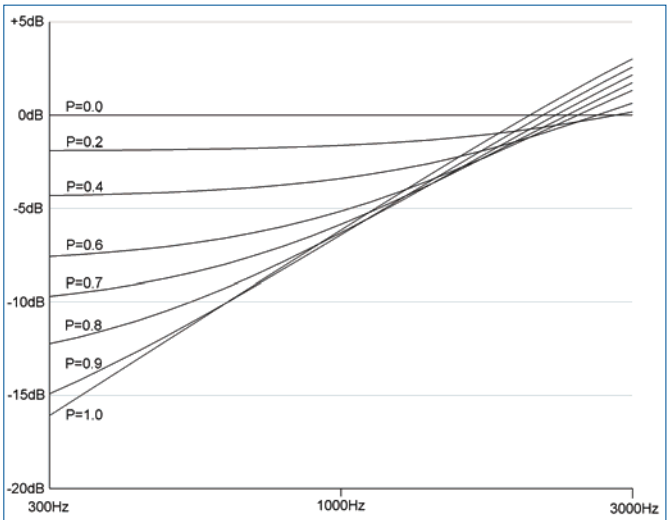


FIGURE 1: Pre-emphasis response curves.

INTRODUCTION. In this final part we cover the transmitter software, from the microphone input to the point where the digital data output from the computer is fed along the USB cable to the RF unit for upconversion to the transmit frequency. The last paragraphs describe how the various digital parts of the project synchronise to each other and to other digital systems.

But first, a topic left over from the earlier parts. In the RF unit there is a step-attenuator between the receive antenna input and the ADC, which has 16 x 1.5dB steps controlled from the computer via the USB. The ADC has an auxiliary 'overload' output pin, arranged to light a LED on the board. In an analogue receiver the distortion products increase smoothly with input level so we keep the gain low but just high enough to hear the antenna noise. There is no such 'smooth' distortion in a digital receiver so we keep the gain high but just below the overload point. In the early work it was clear that even if the overload LED flashed occasionally, there was no sign of distortion in the receiver output. This gave us an idea for an automatic process to set the attenuator to the optimum point. This involved adding logic to the RF unit that counts the number of times the ADC overloads. A process in the computer uses this result to adjust the attenuator up or down to keep the overload count somewhere between zero and the level at which distortion appears. This process is quite unobtrusive in normal use but the attenuator can also be controlled manually.

SPEECH PROCESSING. The microphone input would normally come from a sound card, either plugged into the computer or built into it. The frequency response of this input will be flat but for communication purposes a rising response is often preferred. Special microphones are available that achieve this but we can do it in DSP very easily. If we subtract a chosen fraction (P) of the previous input sample from the current sample, the result is a frequency response that is flat when P=0.0 and rises at 6dB/octave when P=1.00. **Figure 1** shows the response of this process, plotted over a range of values of P. Good reports have been received with P=0.6, using a 'flat' microphone.

To cater for variations in microphone level, an automatic gain control is included. The same 'divide by the peak' process, used in the receiver AGC, is employed but with a much longer time-constant.

Human speech is quite 'peaky' and it's not easy to handle it efficiently with practical transmitters. However, clipping the peaks can be an unsatisfactory way to improve the 'talk-power', introducing significant audible distortion. Also, in the case of an SSB transmission, the clipped audio waveform from such a clipper does not produce a clipped SSB envelope, so most of the benefit of clipping is lost. A variety of techniques have evolved over the years to overcome these problems, but in the analogue world these techniques can be expensive and usually involve processing the modulated SSB signal rather than the speech itself. In this DSP design, a much simpler solution is proposed that does process the speech itself and is therefore usable for other modulation methods, not just SSB.

If we form a control signal from the instantaneous magnitude of the audio input and divide the input by this if it exceeds a threshold level, the result will be the same as if we hard-clip the audio. This isn't a solution as it stands but if we could ensure that the control signal contained no components at the signal frequency or its harmonics, this would guarantee that the controlled output was free of harmonic distortion. In part 2 we used Pythagoras' theorem to form the envelope of a received I/Q signal to ensure that the input signal didn't appear in the output of the AM demodulator, and this is exactly what we need here. If the microphone audio was available in I/Q form, we could make a signal-frequency-free magnitude detector and use it to make a harmonic-free clipper.

There is a DSP function called a Hilbert transform that, for our purposes, is a broadband 90° phase shift network. With a sine wave input, the output is a cosine wave. Such a device can be used to convert an audio waveform into I/Q form. If we label the input to the Hilbert transform as 'I', then its output can be labelled 'Q'. Calculating the square-root of the sum of the squares of I and Q will give us the control signal we need. The finished DSP speech processor then consists of a Hilbert transform block (a kind of all-pass delay-line filter), the 'hypotenuse' calculation, and a division operation. Apart from a low-frequency limitation related to the length of the delay line, this gives us a broadband, harmonic-free clipper. The flowchart of a speech processor using this technique is shown in **Figure 2**. Note that since the Hilbert transform introduces a delay to the 'Q' signal, an equal delay is applied to the 'I' signal by tapping halfway along the delay line. Good results can be obtained with about 14 – 20dB of clipping. **Figure 3** shows a comparison between a conventional clipper (in red, shown inverted for clarity) and this speech processor. The input, in both cases, is a sine wave rising linearly to twice the clipper threshold.

THE MODULATION PROCESSES. In the earlier parts of this article, the concept of a zero intermediate frequency signal was introduced for the receive path. The same concept applies to the transmit path but we don't need a wide bandwidth or a high dynamic range. We can manage with a sample rate of 12kHz and 16-bit signed digital data for the I and Q signals. The job of the transmit software is to transform the microphone audio to zero-IF I/Q data with the chosen modulation and send this along the USB cable to the RF unit, from where it

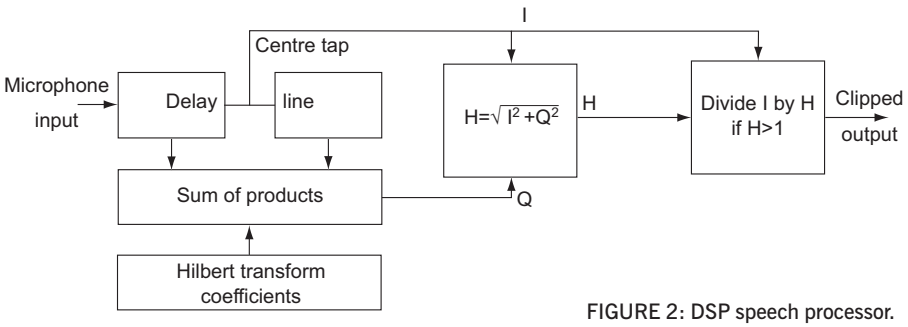


FIGURE 2: DSP speech processor.

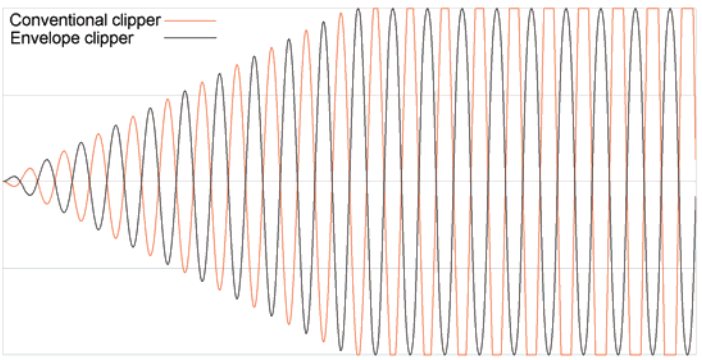


FIGURE 3: Comparing conventional and harmonic-free envelope clippers.

will be up-converted to become a band of modulated RF centred on the chosen 'dial' frequency.

The SSB modulation process needed to convert the microphone audio to I/Q form is essentially the reverse of the receiver process outlined in part 2. In the transmit version, which is similar to the DSP phasing generator described in **[1]**, the audio input is fed along a tapped delay line. Two sum-of-product processes implement a pair of 300 – 3000Hz bandpass filters with identical amplitude responses but phase responses that differ by 90°. One filter output becomes the I-channel signal to be fed to the RF unit and the other becomes the Q-channel. When combined in the RF unit, one sideband adds and the other cancels, in the same way as in a traditional phasing-type SSB generator.

For AM and FM we must remember that the audio path up to this point is broadband, so the first item on the agenda is a 3kHz low-pass filter. In a traditional transmitter this would be preceded by a clipper to guard against overmodulation, but the speech processor already described does this better, so an overmodulation clipper is not needed. To generate AM we just scale the signal to swing over the range 0.0 to +1.0, and that becomes the I-channel output to the RF unit. The Q-channel stays at zero. The final transmitter output will be an AM modulated carrier at a fixed phase on the centre frequency.

For FM modulation, the standard 750μs pre-emphasis is applied first followed by the speech processor and the low-pass filter. The signal is scaled so that the clipped peaks are equal to the desired deviation divided by the samplerate. To convert from frequency to phase, we sum this, on each sample, into a 'cycle accumulator', discarding whole cycles and keeping only the fractional part. This is multiplied by 2π to give the desired carrier phase Φ in radians. To generate a carrier at this phase as a zero-IF signal, we form $I=\cos(\Phi)$ and $Q=\sin(\Phi)$ and these go straight to the RF unit as an I/Q pair. The final transmitter output will be a constant-amplitude carrier swinging either side of the centre frequency.

In part 2 we described how CW reception was arranged so that the centre of the received passband was exactly on the dial frequency. To ensure that the transmitted CW frequency is the same, we need to generate our software CW signal in the centre of our zero IF. A keyed DC signal followed by a key-click filter feeds straight to the I channel. CW enthusiasts would probably expect to have a key-jack provided, but so far only a software Morse keyboard sender has been used.

SYNCHRONISING THE ADT HARDWARE TO THE COMPUTER.

The ADT hardware is configured so that the computer 'thinks' it's a USB soundcard. The USB specifications define that all data passing along the bus is timed from a 1kHz clock supplied from the computer. A conventional USB sound card will typically derive its sample rate timing from this clock by means of a phase locked loop, but in our application we really want to use the clock in the ADT hardware to generate the timing for the whole system. Our clock is far more accurate and stable than anything else in the computer.

A conventional USB sound card handles 48kHz input by sending exactly 48 samples down the USB on every 1kHz clock pulse. Because there may be a tolerance offset between the ADT sample-clock and the USB clock, the USB chip on the ADT board may need to send 47 or 49 occasionally. The USB specification handles this without problems. The data arriving in the software is thus in blocks of uneven size, but every sample from the ADT board is delivered to the software.

In the transmit direction the same process occurs, with the number of samples per USB clock cycle being modified from the nominal figure (12), but in this case it's the ADT hardware that has to decide how many samples it wants the *computer* to send on each clock pulse. The USB specification defines a 'flow-control' signal to do this, which the USB chip on the ADT board sends. Again, the data arrives at the ADT board in blocks of uneven size but every sample from the software is transmitted on time.

SYNCHRONISING THE SOFTWARE TO THE SOUNDCARD.

A standard computer sound card input consists of an analogue to digital converter (ADC), which is clocked by a hardware timer that determines the rate at which digital samples are captured and sent to the associated software. The operating system and the application software process this data, for example to record incoming audio to a file. Although the software may well handle the data in blocks or buffers, the mean rate at which it must do so is ultimately set by the hardware. A computer sound card output likewise has a digital-to-analogue converter (DAC), also clocked by a hardware timer, and any software that is 'playing-back' an audio signal, for example from a file, must feed data to the DAC at the rate set by the hardware.

This works fine for simple record-to-file and playback-from-file tasks, and even audio pass-through applications will work successfully without data overflow or underflow if we know that the ADC and DAC clocks are derived from the same source and will not drift relative to each other.

Our receiver is effectively a 'pass-through' process but with different clocks, one in the ADT hardware and the other in the speaker soundcard. There will inevitably be a tolerance error between these two and we do not have the ability to lock one of them to the other. We must solve this one somehow or risk corruption of the speaker audio.

The solution involves re-sampling the audio data stream as it comes from our software (at a sample rate of precisely 12kHz, derived from the ADT hardware), re-timing it to the actual sample rate of the speaker soundcard. To visualise how this is done, imagine first that we just want to delay an incoming audio stream by a fixed fraction of a sample interval, to feed an output at the same sample rate. Instead of simply copying the delayed input sample across to the output, we will do a linear interpolation between the preceding and following input sample values. For example, if we choose the output to be one-third of the way along the input sample interval, we add 2/3 of the

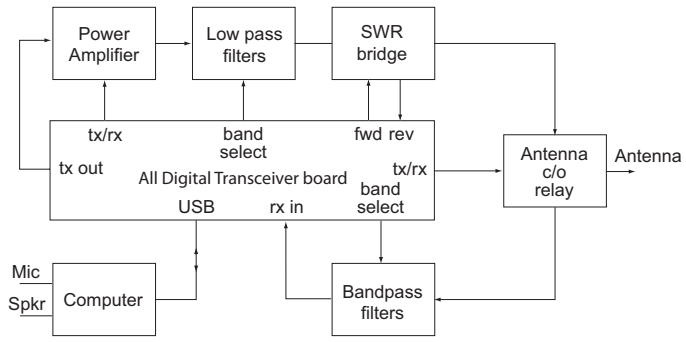


FIGURE 4: Block diagram of a complete transceiver based on the ADT.

preceding sample to 1/3 of the following one. Now imagine that we want to vary the delay fraction between zero and one. We just vary the proportions of the two adjacent input samples that we use to calculate the desired output value.

Now suppose we move the chosen output sample point continuously in one direction, and continuously update the interpolation equation as the chosen output point slides between and through the input samples. The output will always be a smoothly interpolated version of the input but at the offset sample rate. To finish the job we need to know in which direction to move the interpolation point and how fast to move it. This is done by examining the state of the output sound card buffer, sliding the chosen output point one way or the other in response to the amount of data in the buffer, aiming to keep this buffer approximately half full.

This simple linear interpolation introduces a slight modulation of the high-frequency response as the interpolation point varies but this can be corrected by calculating the output value from a run of 3 input samples instead of 2.

A similar synchronisation problem exists in the transmit direction, since the microphone audio comes from the computer sound card and the transmit data is clocked by the hardware in the ADT board. Another retiming process deals with this by monitoring the state of the microphone input sound card buffer and adjusting the rate at which microphone samples are converted to transmitter samples.

SYNCHRONISING BETWEEN OUR SOFTWARE AND OTHER SOFTWARE.

The above retiming process is sufficient for listening to speech and CW, but we might want to feed the audio output to some other software, for example to demodulate some digital data or analyse the received spectrum. If we were to do this with a conventional analogue receiver, we would do it with an audio cable from the auxiliary output of the receiver to a sound card input.
We could do the same here but somehow it doesn't seem right to be linking two digital audio programs with an analogue interface.

One popular way of linking two digital audio programs in this way is to use a software device known as a virtual audio cable (VAC). Such a device behaves like a sound card output linked digitally to a sound card input, but there is no physical hardware and the signal path is entirely digital. In use, one program will be run with a VAC selected as its output, and the other program is run with the other end of the VAC selected as its input. Since all such programs must derive their timing from the selected soundcard, the VAC software *itself* must supply this timing to both ends. This is derived from a hardware timer in the computer. The tolerance error of this timer doesn't cause any problem where a VAC is used to link a playback-from-file program to a record-to-file program – the data is transferred accurately even if it takes slightly more or less time to transfer it.

If we route the output of the ADT receive software to one end of a VAC instead of to a speaker, the tolerance error of the VAC timer is handled by the retiming process already described, but the program on the other end of the VAC will experience the VAC timer tolerance error. For example, a frequency-measuring program would not read modulation tone frequencies accurately. This also means errors in the data rates of digimode programs. Some digimode programs have the ability to calibrate-out the sample rate tolerance error, and in a lot of cases the error may not be so bad that it causes trouble, but it's worth discussing here because there is a better way. Since the clock in the digital receiver is the same one that determines the accuracy of the RF frequencies, and this is by far the most accurate source in the entire chain, it's worth trying to maintain this accuracy throughout the chain. The virtual audio cable method will not do this.

What is needed is a 'virtual' sound card interface in the digital radio software itself, which although it doesn't exist physically as a soundcard, behaves like one. The 'other' software should 'plug into' the digital radio software 'soundcard' rather than having both programs plug into a virtual back-to-back connector. We have created such an interface, which installs into Windows as a device driver, and streams audio to and from our software when it's running. The 'virtual sample rate' of this interface is determined by the digital radio software, in turn determined by the digital radio hardware, not by the rather less accurate crystal in the computer sound card, nor by the timer used by a virtual audio cable device. The virtual sound card also has a matching output port which handles the corresponding transmit-from-software function. The end-user will simply run his favourite digimode program, select our virtual sound card instead of a real one, and be confident that the frequencies and data rates within the program will be accurate.

With the data streams accurately timed from the ADT board right through the entire software path, time and frequency calculations in the software will be as accurate as the reference oscillator in the ADT. This opens up interesting possibilities which would be very difficult to engineer in an analogue design. For example, with the ADT board driven from an external frequency standard locked to a GPS-based source, it's possible to receive a broadcast frequency-standard signal, display it on a software-referenced phase display such as a Lissajous figure, and hold it rock-steady for ever. Transmissions from an ADT driven by a software source will likewise be as stable as the reference input.

CONCLUSION.

Figure 4 shows how a complete transceiver could be constructed using an ADT board. At G3PLX, most of the RF parts were rescued from an old analogue transceiver.

This series of articles has described the authors' all-digital transceiver project, which we have worked on for about three years now. The results are very encouraging and we are sure that the all-digital approach has a place in amateur radio but it does mean learning some new tricks. In these articles we have thrown out some of the traditional analogue signal-processing ideas, such as the superhet and the AGC loop and introduced new ones, in particular the concept of the I/Q signal path. We have also highlighted some of the aspects of the digital approach for which there are no equivalents in the analogue world, such as synchronisation of the data streams. We are both continuing the development of the all-digital approach, but we hope that others may pick up some of these ideas and carry them forward.

APOLOGIES AND THANKS.

In part 1 we expressed the hope that ready-built ADT boards could be made available at a reasonable cost for amateur use, but this has not yet been possible.

In Part 2 there was a typographical error half way down the middle column on page 28. The formula should have read $I*\cos(2*\pi*F*t) + Q*\sin(2*\pi*F*t)$. The π character became corrupted in the production process.

Thanks are due to Andrew Senior, G0TJZ for helpful comments during the preparation of all three parts of this article.

REFERENCE

[1] A digital SSB phasing network, Peter Martinez, G3PLX, RadCom June 2004 p 84.