Synchronous Decoding in WSQCall

Murray Greenman
22 March 2018

Introduction

WSQCall is a newly developed high-performance radio mode for LF and MF. It was designed to provide chat (QSO) capability under very weak-signal conditions, or conditions with deep fading. Until this development, reception with signals weaker than –25 dB SNR was limited to strongly error-corrected beacon modes such as WSPR.

These beacon modes have fixed, precisely timed transmission frames, and are highly structured, with strong error correction. However, such modes only ever send fixed data and are very slow. No possibility to send interesting information or ask questions. Useful for what they do well, but with no QSO capability at all!

WSQCall[1] came together through a series of curious and fortuitous circumstances in the long-running ZL2AFP/ZL1BPU cooperative development saga (since about 2003). First the original DominoEX, WSQ2, then FSQ and FSQCall, FSE (synchronous decoding of error corrected files), and finally WSQCall came along, all IFK+ modes.[2] All these had a part to play in the development of the Synchronous Decoding algorithms now used in WSQCall V1.20.

This paper describes how the Synchronous Decoding process works.[3] There will soon be another paper that describes how the Selective Calling business works.

Decoding Methods

Modes such as WSQ and FSQ transmit phase continuous tones with no transmitted synchronising information, so symbol[4] synchronism needs to be recovered from the data stream. The data is transmitted as incrementally coded FSK using 33 close-spaced tones.

The WSQCall receiving process consists of a series of overlapping Fast Fourier Transform (FFT) processes, operating with a 12 kHz sampling rate from the sound card, decimated to a lower data rate. There are 16 FFTs run per transmitted symbol (at ~0.5 baud), resulting in 16 detector results (reporting the bin number with the most energy) per symbol. So there are about eight solutions per second.

Other speeds are offered (1 baud, 0.25 baud), and these are achieved by ‘changing gear’, changing when the sound card data is sampled (using decimation). This has the advantage that no other code needs to be changed when the speed is changed, as everything is timed by the sound card sampling rate.

The ‘conventional’ decoder in WSQCall and FSQCall uses a simple synchronisation mechanism called a ‘peak hits’ decoder. This was first conceived by Alberto I2PHD, and demonstrated in JASON. In essence, the decoder inspects the FFT samples (reporting the bin with the most energy),

[1] The name means *Weak Signal QSO with Selective Calling*.
[2] IFK+ means incrementally keyed (differential) MFSK modulation with a fixed offset.
[3] We’re talking here of MFSK data symbol synchronous, not RF carrier synchronous!
[4] The *symbol* is the smallest unique element of a digital transmission, where the phase, frequency and amplitude are for the moment unchanged.

and determines when the bin number has changed. Specifically, any point where the bin number is the same for three consecutive samples is considered a valid symbol, and when it changes another symbol is assumed, and is confirmed over a further series of samples. This is essentially an asynchronous process, and is capable of decoding accurately over a remarkable 6:1 speed range. The main advantage of this is that it makes the decoder completely immune to ionospheric time-of-arrival errors, a very important factor in the performance of FSQCall, which was designed primarily for HF NVIS propagation.

The problem with the 'peak hits' method is that it is prone to errors when the signal into the FFT process is very weak. An improvement was made recently in WSQCall by increasing the FFT overlap, and offering an alternative algorithm that looked for three *cumulative* samples the same rather than three *consecutive* samples the same, but the accuracy still suffered when signals were very weak.

It was clear to us from long observation that when signals were weak the transition from one symbol to the next in the FFT samples was not clean, and was not accurate in time. This is easily observed by plotting the samples on a graph (see below). The implication is that if the data is not clean, there is more room for error, especially in a decoding process such as 'peak hits', which is essentially non-synchronous in nature.

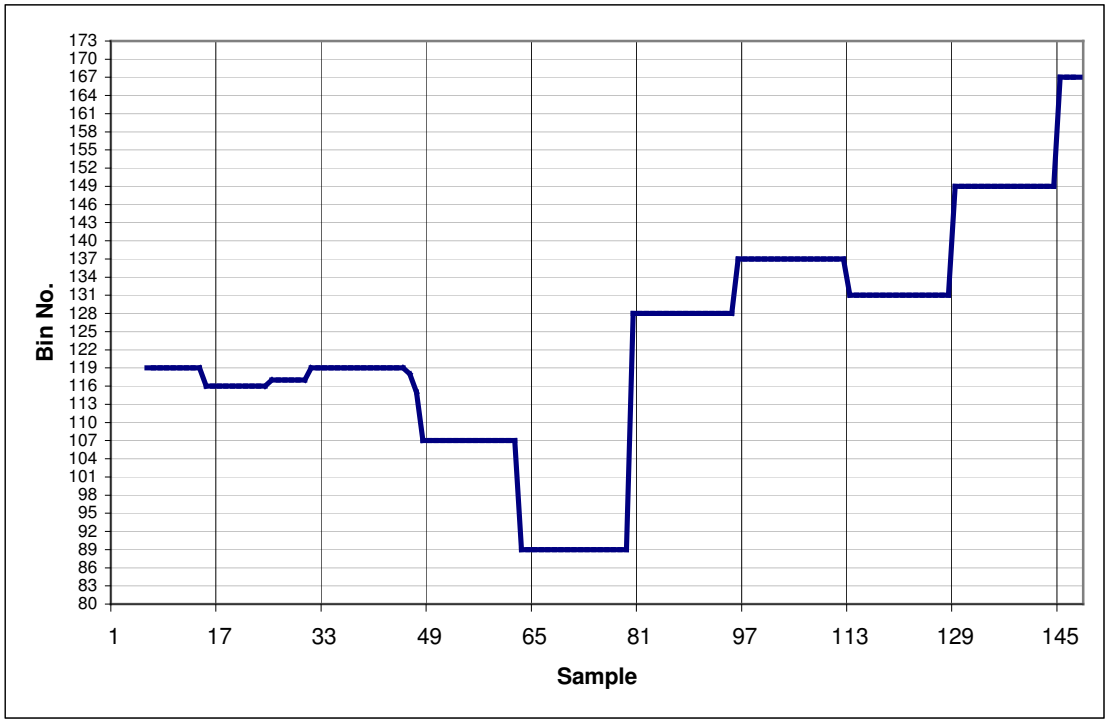

Fig. 1. Real symbol samples plotted

Figure 1. is an Excel graph (essentially frequency vs time) of a small fragment of real WSQCall FFT detector data. The received signal was only slightly noisy. The graph has been arranged so that it has vertical grid lines every 16 samples (the nominal symbol rate), and horizontal grid lines every three bins (the nominal tone spacing). Note how the symbols received don't always line up with the time grid, and that the data isn't always spaced accurately, or constant through each symbol. It's a lot worse than this on really weak signals!

It was at this point that thoughts turned to synchronous sampling and decoding.

Synchronous Sampling

The idea is not by any means new, as clock recovery from an MFSK synchronous data stream was first demonstrated in MFSK16 (1999),[5] where a fast FFT was used to recover a triangle-shaped energy profile from the data stream at 15.625 baud, and used to control a conventional phase-locked loop clock.[6] But applying synchronous sampling to reception of a very slow MFSK signal buried in noise was quite another matter. A phase-locked loop would be much too slow, as experience with MFSK16 showed that it required 20 or so symbols of 'training' to lock on. At 0.5 baud, that's some 40 seconds!

During the development of an error-corrected file transfer facility for FSQCall (FSE, Fast Simple Error Correction), which used Reed-Solomon coding, the author developed a way to recover synchronism from a data stream in a post-processed manner. The overall technique was quite complex, but was needed because the receiver required a way to avoid insertion and deletion errors which are the enemies of reliable FEC decoding. However, the sync recovery scheme used, which involved a cross-correlator, is highly pertinent to this discussion, and the concept proved to be ideal for sync recovery in WSQCall.

One of the major benefits of a synchronous decoding process is that, while errors in content can be made, it is not possible to generate insertion or deletion errors (extra or fewer symbols) in the data stream, and this in itself leads to much better decoding accuracy, as each symbol is decoded on its own merits, not dependent on the history of prior decodes. Of course the process also benefits from accurate sampling in the middle of each symbol, where the energy in the FFT bins is greatest, and samples are more likely to be stable.

The latest WSQCall V1.20 has slightly improved sensitivity over earlier versions (about –27 dB SNR at 0.5 baud), but the quality of the decoded messages is significantly better. Since the output of both synchronous and 'peak hits' decoders is visible to the user, it is easy to compare the results.

Sync History

In the WSQCall receiving process, the FFT samples are written to a buffer, from the time the squelch opens, until it closes again. They are also written to a file, *raw.RS*.

As this sampling process is going on, an interesting graph is drawn. This has no purpose in the decoding process, but nicely illustrates the signal quality to the user, and also forms a useful introduction to the synchronous decoding process.

In real time, as the samples are taken, the bin number represented in each sample is subtracted from the number in the previous sample, and the result squared. If the square is larger than a predefined threshold, then a dot is plotted on a graph, representing that a change in symbol has been discovered.

- *Recall that in WSQCall, the tones are spaced 3/T; three times the baud rate. The FFT also operates with a bin resolution of three times the tone spacing, so a symbol number difference of ±3 (9 when squared) represents the minimum one step between symbols. In an IFK+ modulation scheme, there are* ***never*** *repeated tones.*

[5] *MFSK For the New Millenium*, Murray Greenman ZL1BPU,QST January 2001. Code written in 2000 by Nino Porcino IZ8BLY.

[6] The *only* known examples of MFSK use prior to this *Amateur* development were Piccolo and Coquelet, and both used electronics rather than DSP. Importantly, both also transmitted specific sync information.

The graph starts at the top left corner when squelch opens, and moves down one pixel with each sample. So each column represents one symbol duration. After 16 samples, the column is advanced one pixel, and the pixel position returns to the top. Since there are 16 samples per symbol at the FFT output, this graphic reveals a nice depiction of the synchronism as the signal is received.

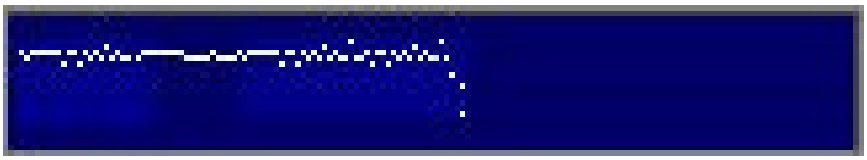

Fig. 2. The Sync History Graph

Figure 2 shows the graph just described for a reasonably strong signal. It's quite clear that the symbol change points are depicted in a more-or-less straight line. The sudden drop at the end represents noise sampled before the squelch closed. This and the next few graphs are real snap-shots taken from on-air signals received by WSQCall V1.20.

Just by eye, you can imagine how, if the data was sampled 8 samples further down this graph, the samples would be well clear of any change-point uncertainty.

Fig. 3. A weak signal history

Now Figure 3 shows a history graph for a much weaker (-27 dB SNR) signal. While there is greater spread of the symbol change points, you can imagine how sampling where there aren't any dots (in this case across at the bottom of the graph) would avoid all the change points.

Another interesting thing that can be noted in Figure 3, if you look really closely, is that there are places where more than one dot appears in a column, or there is no dot at all! The beauty of the cross-correlation sync discovery method about to be described is that it is completely immune to such limitations.

The Cross-Correlator

In order to facilitate development of the Synchronous Decoder, a feature was added to WSQCall (still at this point using only the 'peak-hits' decoder), where all the FFT samples (a list of bin numbers) were written to file. This allowed the author to develop a Synchronous Decoder prototype independent of the main program. Indeed, since the sample file is still available in WSQCall V1.20,[7] anyone who can dream up an alternative-decoding algorithm can still have a go! Can you do better than the decoders in WSQCall?

The prototype was written in QB64,[8] and screen-shots of it will be used later to illustrate the process. The first action is to determine the sync position relative to the start of the file. We analyse the sample data with a cross-correlator, which consists of a 16-bin array, which accumulates symbol changes. Having seen the Sync History graphs in Figures 2 and 3, you will quickly grasp how it works.

The first process is to clear the correlator array, and set the array index to bin one. Then the sample file is examined one sample at a time, just as described for the Sync History, except that now this is

[7] Working folder/Shared/raw.RS

[8] A 32/64 bit BASIC compiler for Windows. The program ESH3.bas 01/03/18, source available from the author

a post-process, undertaken once the reception is complete and the squelch closed. Each sample is compared with the one before, to determine the difference squared, and if the threshold has been reached, the corresponding cell in the correlator array is incremented. Then the array index is incremented and the next sample examined in the same way.

The correlator index is incremented modulo 16, so every 16 samples, it starts at index one again. So the energy in the bins of the correlator array is accumulated synchronously, and will begin to represent where the sync change points have been discovered. By the time the end of the file has been reached, a very reliable picture of where the sync change points were will have been determined.

Fig. 4. Strong signal correlation

Figure 4 shows the result. The height of each bar in the histogram represents the number of change points found at that index in the array. This is for the same data as in Figure 2, and you can see that most of the energy is concentrated in bin 3. It compares well with Figure 2, where most of the dots run along in a line about 3 pixels from the top.

So clearly the symbol change points are every 16 samples, starting three samples from the start of the file. To sample the data for decoding, the place to sample the data (to achieve the cleanest, most reliable and noise-free samples) would be eight samples later, at sample 11 and every 16 samples thereafter.

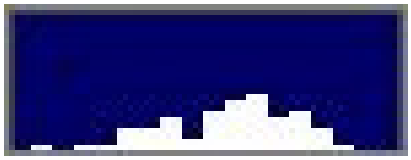

Fig. 5. Weak signal correlation

We call the histograms in Figures 4 and 5 *Correlograms*. They are pictorial representations of where, from a statistical point of view, the symbol change points occur. In Figure 5 (same –27 dB weak signal as Figure 3), the peak is not as clear, but most of the energy is centred on bin 12. The spread represents the uncertainty in the symbol change point on a very weak signal. It's easy to determine that the appropriate sampling start point would be sample 4, eight samples earlier.

It is important to recognise that the cross-correlator is an integrating process, and so is relatively insensitive to noise (wrong or missing change points); and the longer the received sentence, the more secure the peak becomes. The errors are stochastic in nature, while the correlator is a 16-bin integrator that is sensitive to coherent data, but insensitive to asynchronous effects and random noise. This simple cross-correlator is capable of reliably determining the sync position on the very weakest of signals.[9]

A cross-correlator compares one product with another. In this case, one product is the received signal samples, but what is the other? It is the simple known fact that *on average* the symbol change points will occur every 16 samples. If you like, it's the same as comparing the samples to

[9] The bigger problem is getting the Squelch to open and close reliably on weak or fading signals. We looked at 'digital' squelch based on examining the sync samples in real time, but have so far not found a signature which made any improvement over examining the energy in the main FFT. The idea also tended to be slow in response.

‘0000000000000001’ in a sliding manner. That’s still a cross-correlator, even if it’s a very simple one.

Limitations

There are two limitations to the use of the simple cross-correlator as described. These are:

1. The idea works only if you know the exact symbol rate (unlike the ‘peak hits’ method, which has amazing tolerance of timing differences). As a consequence, its results are adversely affected by poor sound-card timing. Nor would its use on HF be appropriate.
2. If one signal ‘doubles’ over the end of another, decoding is severely compromised. If the sync phase of the two signals is similar, a good decode of both might result, but more likely one or the other (even both) will not decode correctly.

This second point will be considered in a moment.

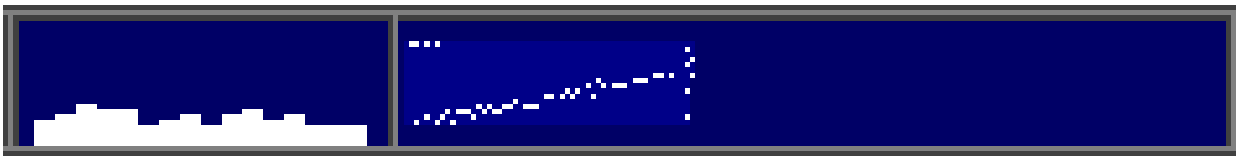

Fig. 6. Poor sound card timing

Figure 6 illustrates what happens if one station has poor sound card timing compared to the other. Unless the sentence transmitted is very short, the symbol change point will change over time so much that the correlator information will be smeared and highly unreliable. In the example illustrated, despite being relatively noise-free, the Synchronous Decoder did not decode the signal, although the ‘peak hits’ decoder came up with a perfect sentence! Note how ‘flat’ the Corellogram is, with no clear peak.

Decoding

Once the optimum sampling point has been determined, the received sample file is simply sampled every 16th sample,[10] and the tone numbers decoded using the IFK+ decoding algorithm. The algorithm subtracts sequential tone numbers to discover the difference, divides the result by three and rounds to remove drift and Doppler, subtracts one from the result to remove the tone rotation, and then adds 32 if necessary to bring the symbol number into the range 0 – 31. The symbol number is then decoded to an alphabet character using the WSQ look-up table. Transmissions always start with two tones representing a ‘space’ character, so the differencing algorithm initially has something to work with.

Prototype Decoder Development

As mentioned above, the Synchronous Decoder was developed as a stand-alone prototype, written in QB64. Several illustrations from this program will now be used to demonstrate its operation.

Please excuse the primitive graphics, but this after all, was a demonstration prototype tool. You have to start somewhere!

[10] We also tried using the mean of three adjacent samples, but a simple single sample proved to be more reliable.

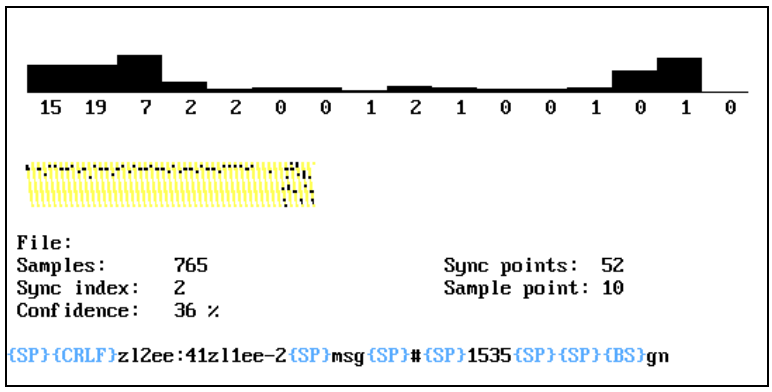

Fig. 7. The prototype Sync Decoder

You can see in Figure 7 the same elements described above, the Corellogram (top) and the Sync History (below it). The numbers under the Corellogram are the change point counts accumulated for each bin. The bin numbers are 1 – 16, left to right.

Then some statistics relating to the received sentence are shown. The 'Confidence' figure expresses the height of the main Corellogram peak as a proportion of all the change points discovered. Finally, the decoded sentence is displayed. Being a prototype, the 'hidden' characters in the transmitted sentence are also shown, displayed in blue, surrounded by braces.

The program determined the symbol change point (2) and the sample point (10), and then decoded the sentence. This is a standard WSQCall protocol sentence, starting with the header, containing a start symbol {CRLF}, a callsign marker ':', source callsign checksum '41', and the destination call sign 'zl1ee-2'.

Then follows a command trigger {SP}, and then the payload message 'msg # 1535 '. The final {SP}{BS} combination is a marker that defines the end of the transmission, and is used to close squelch quickly to avoid trailing rubbish. You can see two characters of trailing rubbish in the above picture.

The prototype followed the exact process just described, and as now used in WSQCall. The only difference is that it was written in QB64 and stand-alone with primitive graphics.

Doubled Signals

Although unlikely to happen often under real conditions, some examples of this problem have been captured. In the first example (Figure 8), the two transmissions had, by some luck, very similar sync phase, and both decoded as though they were one sentence!

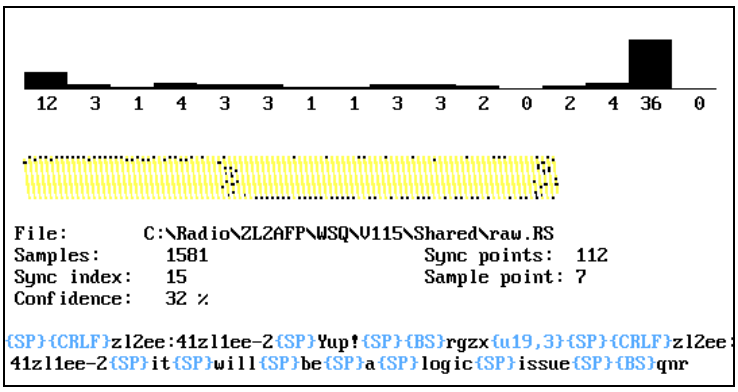

Fig. 8. Two transmissions

It just so happened that the two transmissions in Figure 7 had similar symbol phase. The second transmission closely followed the first, before the squelch had a chance to close, and consequently the receiver continued to record samples until the end of the second transmission.

You can see where (about 1/3 of the way along) the Sync History went noisy, then changed slightly in phase from '1' to '15'. The correlator shows a strong peak in cell 15 and a weaker one in cell 1. The logic determined to sample at point 7, and decoded both messages perfectly. You can see where there were five rubbish characters between the two messages. {u19,3} represents a character code symbol pair with no corresponding character in the symbol table.

The next example shows a very strong signal received, followed by a very weak one. The squelch had no chance to close on such a strong signal (the {SP}{BS} squelch-closing marker was not operating in the receiver at the time).

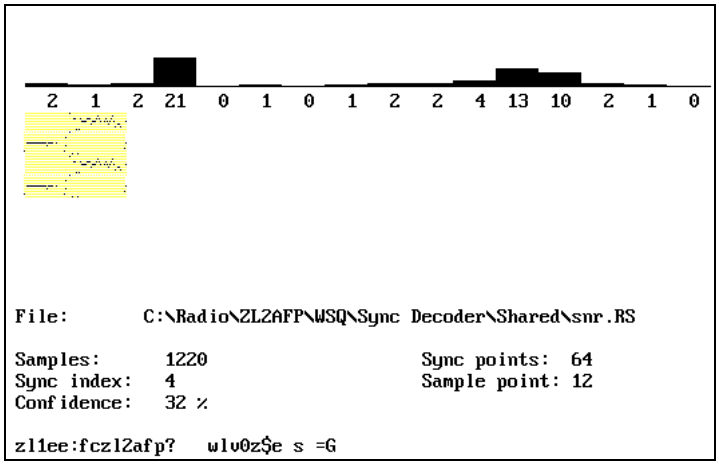

Fig. 9. Two short transmissions, one decoded correctly

In Figure 9, the Sync History shows a marked difference in sync phase between the two short transmissions. The first transmission, very strong, is the one that was decoded, while the weak reply was decoded as rubbish. You can see that the Corellogram shows two peaks. The first one is the stronger, and was the one corresponding to the first signal.

By manipulation of the sample file, it was possible to delete all the samples for the first transmission, and then run the prototype program again, to decode the second message. *This is not possible in the WSQCall program*, but illustrates what can be achieved using an external program. See Figure 10.

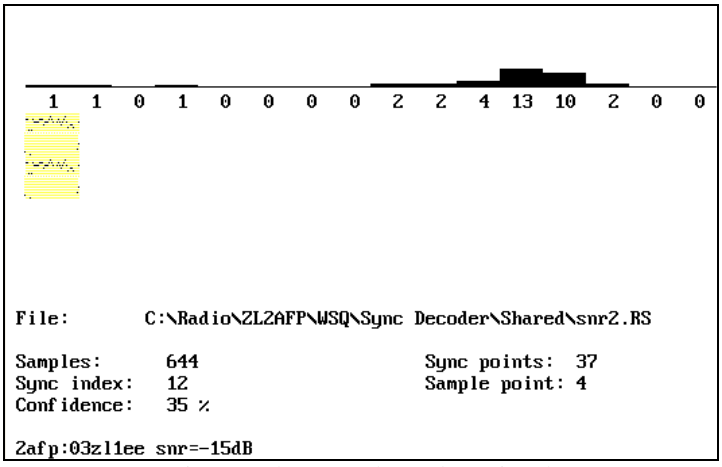

Fig. 10. The second (weaker) signal

Notice the Corellogram peak in Figure 10: it is exactly the same as the right-hand peak in Figure 9, and so it should, as it represents the same samples. Now that the stronger signal at the head of the file has been eliminated, the second weaker transmission decodes. Two characters are missing from the start of the callsign, probably because the transmissions overlapped. But it otherwise decoded perfectly!

It is fortunate that overlapping transmissions are rare, so users will not have this problem too often, given that there is no way within WSQCall to recover two messages. But this example does illustrate what can be done in post-processing the data.

The demonstration program used here is able to capture the *raw.RS* file data as soon as it appears, deletes the file, decodes and displays the result, and then waits for another sample file to appear.

Results

A standardised and repeatable method has been developed for measuring quality of reception using an Ionospheric Simulator, that is independent of the mode under test.[11] The measurement '% Copy' is the number of complete words received without blemish as a proportion of the number of words transmitted, expressed as a percentage. Experience shows that comfortable QSOs can be maintained anywhere above 80% copy, and that reception is very rough below 50%.

WSQCall has been tested in this way, using the AE4JY Pathsim simulator, measuring accuracy of reception against signal SNR. Both the Peak Hits and Synchronous detectors were studied.

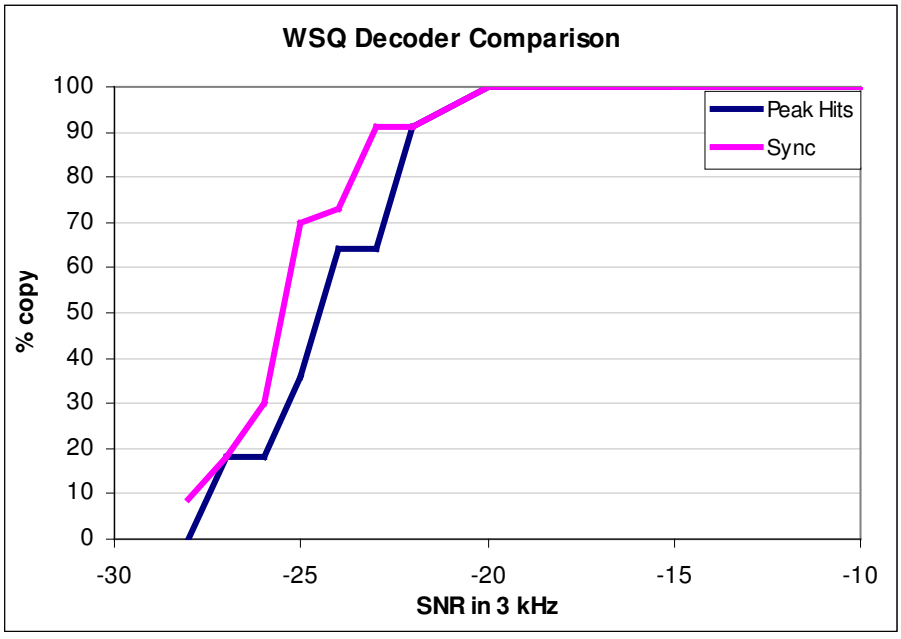

Fig. 11. Performance comparison of signals in noise

Looking at Figure 11, there doesn't seem to be much difference, unless you look across the line at 80% copy, where you see a clear advantage of 2dB, which is only attributable to the reduced effect of noise on the Synchronous Decoder.

[11] See my paper *Ionospheric Performance of FSQ* for details.

Here are some real text examples from the simulator tests at –25 dB SNR:

Transmitted text:	**The quick brown fox jumps over the lazy dog 12345**
Peak Hits Decoder:	**qumk brown fox jumps over the law khg 123ose** (50%)
Sync Decoder:	**jtde quwvk brown fox jumps over the lazy dog** (70%)

Summary

WSQCALv120, with the new Synchronous Decoder, offers a useful improvement in reception quality of weak signals, although little actual improvement in sensitivity.[12] This, and other improvements, some of which also improved the 'peak hits' decoder, has made WSQCall a very useful tool for LF/MF weak-signal operation.

Although not a big issue, the Synchronous Decoder is sensitive to errors or changes in signal timing. Given that the 'peak hits' decoder (insensitive to timing issues) is retained to provide real-time monitoring of the incoming signal, this is not considered a hindrance.

The message parser[13] and the command processor[14] in WSQCall V1.20 only work on the Synchronous Decoder output.

[12] The sensitivity improvements were already available in V1.15.
[13] Locates and validates the source callsign, checks the destination callsign, and determines the trigger command character.
[14] Actions any commands received in an appropriately addressed sentence.