70 Years of Amateur Digital Modes

Murray Greenman ZL1BPU
May 2018
V1.3

Preamble

Amateur Radio operators have been working with digital modes since just after World War 2. Initially this involved adaptation of commercial and military equipment, but from about 1990, new, completely Amateur-developed modes arrived. It is the history of some of this development that this paper describes.

This is also a personal journey, as it documents the path I took from exploring early digital modes to developing groundbreaking and advanced new modes, or at least influencing their development through my own ideas and prototype code.

The order in which the modes are presented roughly matches chronologically my involvement in their development. There are quite a few modes not described here (AMTOR, PACTOR, CLOVER, OLIVIA etc), largely because I wasn't directly involved.

I also cover some developments that aren't exactly digital modes (such as GPS stuff), but are included because they of interest to hams, and involved cooperation with other hams.

1. The Mechanical Age

From around 1950, amateurs began to experiment with ex-military and ex-telecommunications service equipment, typically old (and frequently worn-out) teleprinters. This equipment could be difficult to find, and also it needed to be supported by a significant investment in engineering tools and techniques, maintenance manuals, power supplies, modulating and demodulating equipment, and other mechanical devices such as paper tape readers and punches.

RTTY (Radio Teletype) was the most popular mode in those days, but there was some interest in Hellschreiber (especially in Germany and the Netherlands), and in Facsimile (predominantly in the USA). There will be more about these modes later.

The main problem with mechanical RTTY equipment was that it was landline gear: never really intended for use by radio. Some military RTTY radio networks existed towards the end of WW2, but the equipment used to modulate and demodulate the signals was both cumbersome and scarce. Hellschreiber and FAX were better in that respect, and worked well with existing CW or AM equipment of the time, although again, both modes evolved initially for landline use.

Throughout the short history of digital modes, amateur users have for the most part misunderstood or completely ignored the nature of the medium through which they were attempting to transmit and receive. Indeed, it's still the case today. RTTY is still popular (for contests especially), and yet it is hard to think of another mode less suited to HF DX! This aspect will become more apparent as each of the new amateur-developed modes is introduced.

Handbooks dedicated to RTTY or specialised modes appeared from the early 1960s.[1] Since the first sound-card software, the interest in digital modes has become almost a mainstream interest.

Typical Mechanical Age Set-up

Let's first describe a typical RTTY set-up (mine) from the early 1970s, during the heyday of mechanical teleprinters. I had an old Creed 7B teleprinter, an ex-military version with a 24V motor. It was designed for 50 bps operation. So I needed a 24 V, 5 A power supply just to run it. It generated an immense amount of electrical noise on HF. I also needed an 80 V 50 mA power supply to operate the magnets. With this arrangement, I could actually talk to myself!

The Creed 7B had a fixed type head which spun around, and a paper carriage which zipped back and forth.

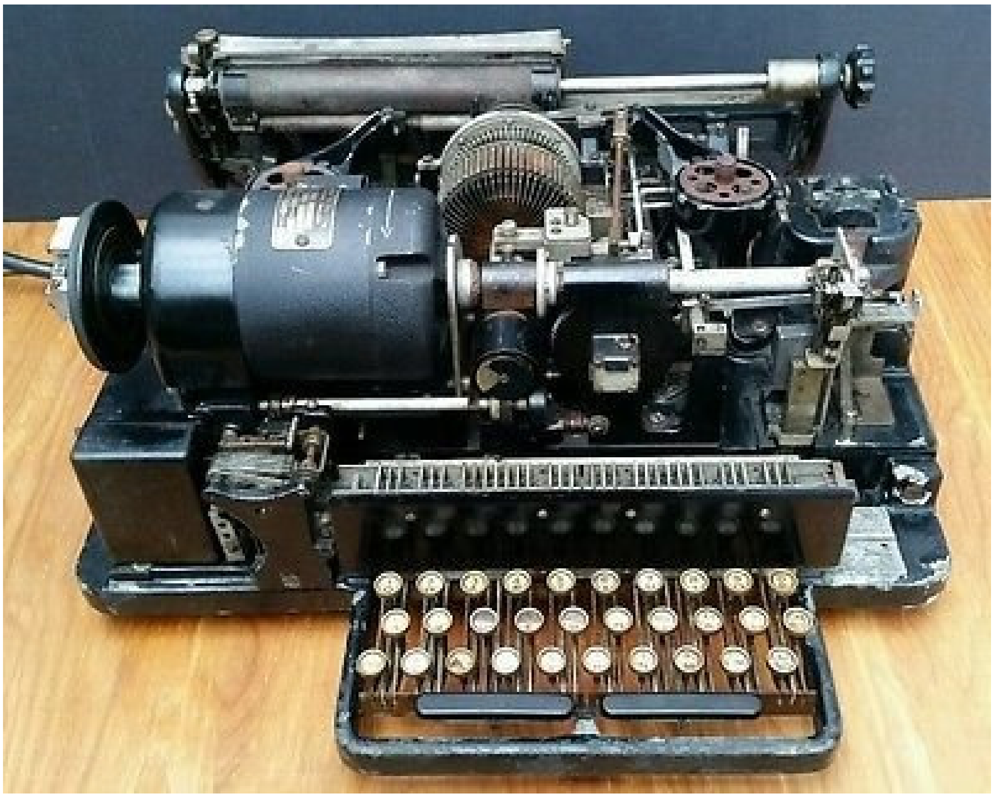


Fig. 1. The Creed 7B Teleprinter

The next step was to find a way to generate a radio signal, and to decode the incoming radio signals. In the early days in the USA, FSK was not permitted, and they used on-off keying, which proved very unsatisfactory due to fading. Later, a number of FSK modems were developed, and a pair of tone standards was settled on: 2125 and 2975 Hz (850 Hz shift), then 2125 and 2295 Hz (170 Hz shift). The former was widely used on VHF, the latter on HF, where selective fading was quite a problem. Commercial stations mostly used 850 Hz shift.

Irv Hoff W6FFC wrote a series of influential articles, published in QST, Ham Radio Magazine and the RTTY Journal from about 1965. Among these articles were several RTTY terminal unit designs. I built an adaptation of the ST-5 demodulator[2] in the early 1970s. This was the first of the 'Mainline' solid-state demodulators.

[1] The oldest I have the *New RTTY Handbook*, Cowan Publishing, Byron Kretzman W2JTP, 1962.

[2] *The Mainline ST-5 RTTY Demodulator*, Ham Radio Magazine, Sept 1970.

I had to change to polar keying (using two transistors) to suit the Creed selector magnets, and also included a crude modulator (FSK keyed audio oscillator) as well.

I also included circuitry to invert the signal (some stations transmitted inverted), and included ‘Autostart’, a system that started the teleprinter motor when a signal was received, and stopped it after the signal was lost. This really was advanced stuff in the early 1970s!

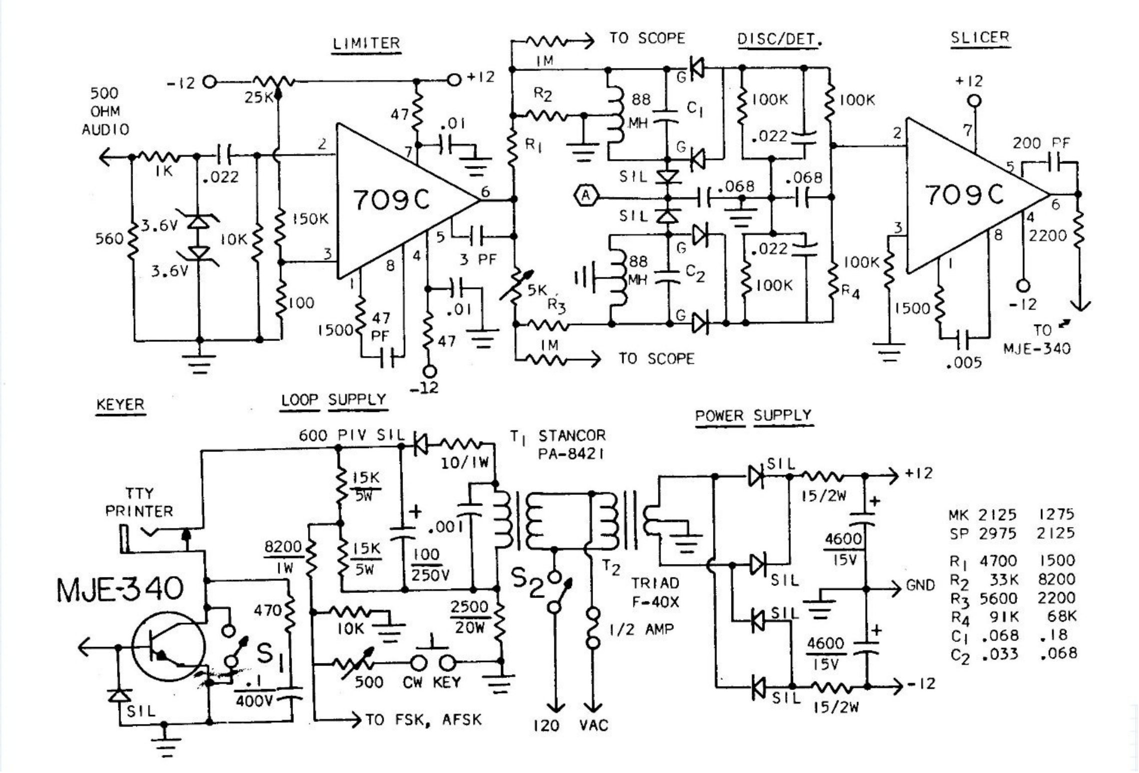
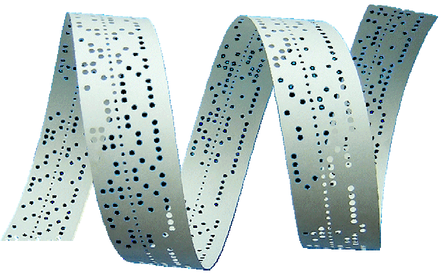


Fig. 2. The original ST-5 demodulator, RTTY Journal, May 1970

I first used my set-up on 144 MHz AM and FM, and worked some decent DX (200 km!). I even held the NZ VHF RTTY distance record for about 20 minutes, until stolen away by ZL1TNS! I also received plenty of commercial and amateur RTTY traffic on HF. In 1983 I graduated to a Grade 1 licence, and started operating RTTY on HF. Results were mediocre to say the least, and I started to wonder why strong signals would not decode reliably, even if accurately tuned. I knew nothing about the ionosphere at the time.

By this time I had acquired a lovely Siemens ‘chadless’ printing tape punch, and a Creed tape reader, which I adapted to read the Siemens tapes. So as well as sending live from the keyboard, I was able to send ‘brag tapes’ and ‘CQ tapes’. I thought I was made!

I later also acquired a Teletype Model 15 teleprinter. This American unit was 110V operated (used a 50 Hz synchronous motor), and had a ‘type basket’, like a typewriter, but which travelled across the paper, fascinating to watch, but very noisy!



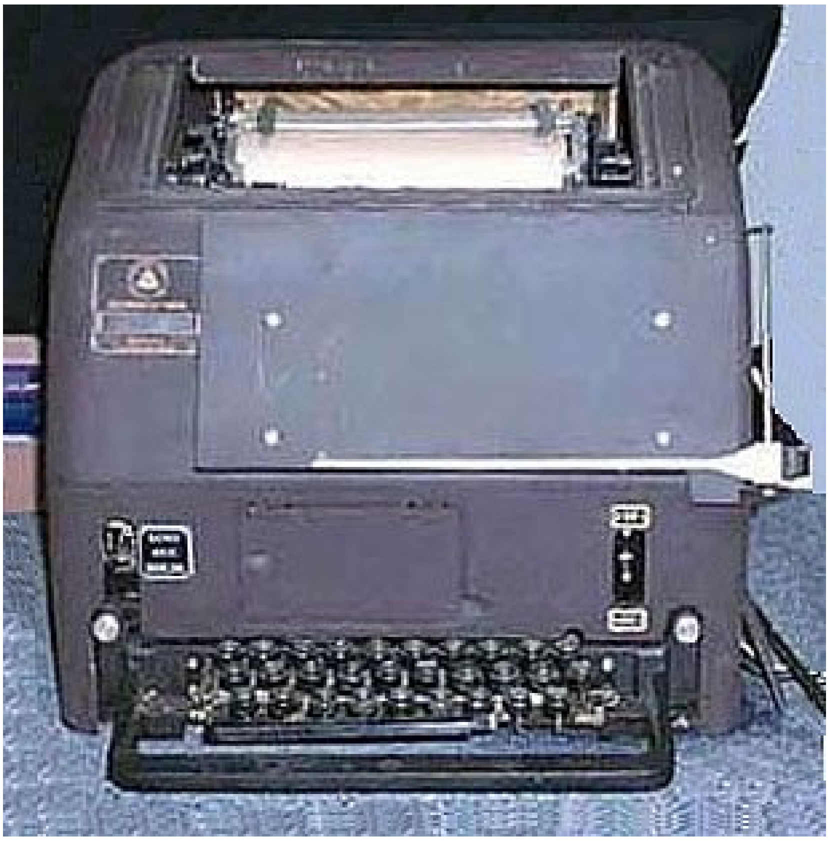


Fig. 3. The Teletype Model 15

One of the problems we had in those days was that teleprinters used motors that were governed at a specific speed, which needed to be very accurate to allow correct operation. Most commercial traffic was at 50 bps, most amateur traffic at 45.45 bps, and given the time it took to change and calibrate the speed, the best solution was to use two interchangeable governors (centrifugal devices fixed to the end of the motor shaft), and change governors to change speed. But that still took several minutes to achieve. Those of us with plenty of money and enough space (not me) simply used two or more machines.

Early Electronics

The best answer was to build an electronic speed changer, to receive 45.45 bps and change it to 50 bps, with the teleprinter equipment all running at 50 bps. There were published designs using shift registers to change speed by the late 1970s, so I started on this route using whatever parts I could find. Most of the unit was TTL chips (about 50 of them), assembled upside down and wired pin to pin, plus an expensive 1024 x 1 bit CMOS shift register in a socket. I had to switch the direction and speed change back down to 45.45 bps in order to transmit, but it worked. The addition of the shift register allowed me to also ‘record’ short messages digitally, to be played back at will. Such sophistication!

The terminal unit consisted of a limiter circuit, a discriminator (using ex-Post Office 88 mH toroidal chokes), a ‘slicer’ (comparator with DC restoration) and a solid-state magnet keyer, generally following the design in Figure 2. The transmit section used a simple audio LC oscillator, the frequency being shifted by switching capacitors.

By about 1975 home computers started to make an appearance in RTTY circles. I remember in the early 1980s Dick Bolton ZL1FL ran an automated station on 3545 kHz for a while – you could send it a message, that it would print, and then it would reply!

But don't forget, all this equipment was large, very noisy, and took up most of my garage (the car – Ford Cortina Mk 2 - stayed outside). Actually completing a conversation with other stations required considerable skill and much effort – just keeping the other station tuned in on my valve transceiver (Yaesu FTDX-400) was an ongoing problem. We should never underestimate the complexity of the mechanical machines, nor the amount of effort that went into putting together and maintaining an RTTY set-up.

Early Facsimile

During the mechanical era I had probably only two FAX QSOs, one with Eric ZL1BDO, the other with Bob ZL1BAD, now both deceased. This would have been about 1985. I recall I used a Xerox Telecopier 400 machine, which burned pictures into carbon paper sheets with a china-clay coating. A horrible smell!

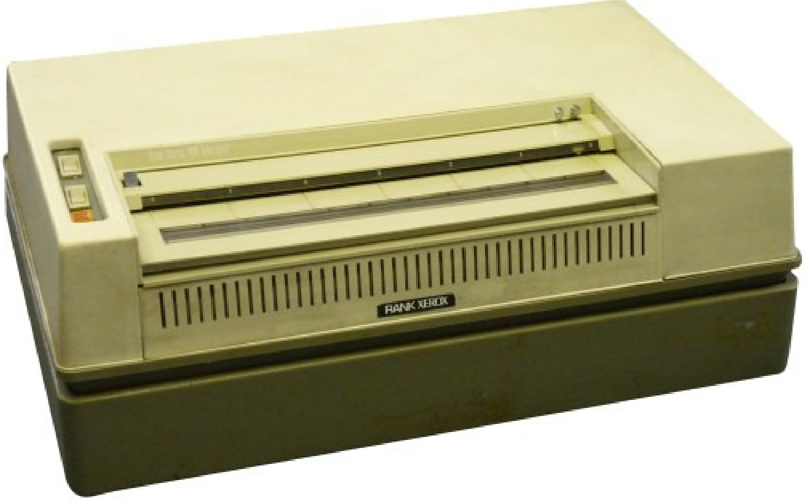


Fig. 4. Xerox 400 facsimile machine

The Xerox 400 was a landline machine that operated with FSK signals, and required no modification at all to operate by radio.

At one point I had a very old valve-operated Western Union Telefax machine, which used AM modulation, but I never used it for a QSO, because I had almost no paper for it. It used silver-coated carbon paper and was very primitive.

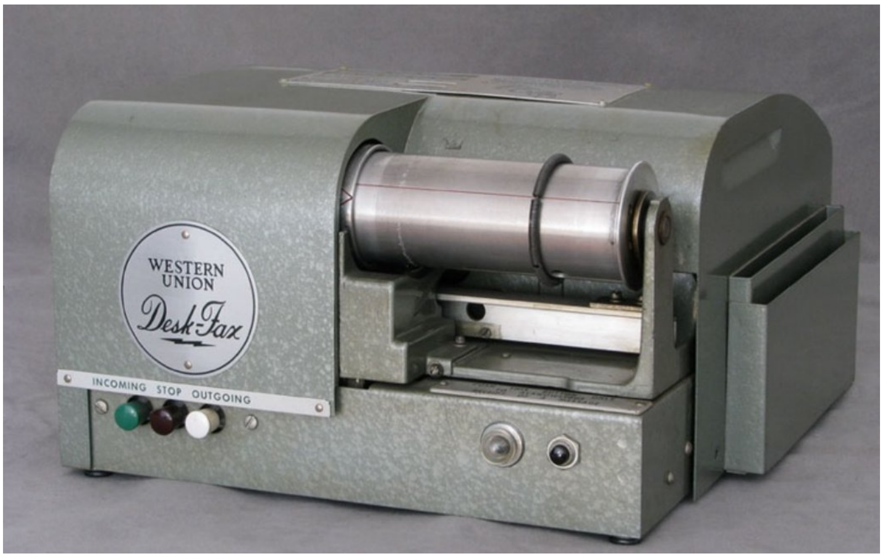


Fig. 5. Western Union Desk-Fax machine

2. The Computer Era

We have already seen how electronics for terminal units, speed changers and memories have evolved. But now the electronics started to have a different purpose – it was interfaced to a primitive computer, which was used to decode and generate signals, and the noisy teleprinter was no longer needed.

Home Computers Arrive

My first serious computer was a UK-101 (very like the Ohio C1P), which I acquired second-hand around 1983. It had a 6502 8-bit processor running at 1 MHz. The display was an old TV monitor. I adapted my RTTY terminal unit to provide a digital interface, and used software written in BASIC and loaded from cassette tape. While this was rather tedious, the change to using a computer was a significant advance – no need for noisy mechanical machines, no need for reels of paper and tape, easy speed changing, and messages could be stored and played from memory.

A few years later I borrowed an IBM PC clone, which had two floppy drives, and I was for the first time able to operate RTTY at almost a moment's notice. In 1986 I moved up to an 8MHz IBM AT clone (all the right chips and software, but no shielding for radio interference), and it had 56 MB of RAM, EGA colour and a 5 MB hard drive! Wow! It cost rather more at the time than you'd pay these days for a top modern computer.

I built a FAX interface for that computer, but more about that later. I also modified my terminal unit to provide a 'Hamcomm' interface, which allowed me to receive weather satellites and other modes (including RTTY) for which DOS software was then available.

Hamcomm was probably the first software demodulator concept widely used. The receiving hardware was a simple comparator, which operated the handshake lines of an RS232 interface. There was no transmit capability, and without filters, reception quality on HF was rather variable. There was software for quite a few modes, including RTTY and FAX.

One of the nicest ham programs around at the time also made use of the Hamcomm interface for some functions, but still used the conventional RTTY decoder. BMKMulty was DOS software developed by Mike Kerry G4BMK, which I bought in 1984 and updated later. It offered RTTY, AMTOR and also PACTOR 1 for the first time. The RTTY mode decoder was the best I'd come across at the time, and the AMTOR mode was also excellent. The tuning display was operated by the Hamcomm interface.

The first of the packet terminals with HF modes were available at around this time, but were rather expensive. I continued to use my hardware Terminal Unit and Hamcomm and BMKMulty software.

Microprocessors

I had also dipped my toe into the packet radio waters, with a fancy new Kantronics KPC-2. This was my first device with a dedicated microprocessor, although I had dabbled with micros and writing assembly code since the late 1970s. At this time (1987) packet radio on VHF was the greatest new thing, and for the first time I was able to send messages around the world, although delivery was slow (up to a week!) and uncertain.

Fig. 6. The Kantronics KPC-2

From about 1989, the Baycom Packet Modem became very popular for a while. It used hardware modulator and demodulator chips to provide a standard 300 bps signal. It ran with DOS software to generate and decode the signals and provide the AX25 packet protocol. Quite a sophisticated and inexpensive system for the time, and it became a further step on the path to software defined reception and transmission.

In 1990 I was made redundant, and went off to the UK to find a job. Of course I took my HF radio and VHF handheld with me, and the KPC-2. No room for anything else! I bought a 386 computer (with Windows 3.1) and quickly got onto the packet network there, and was thus able to send messages home. My family soon followed me to the UK, and we set up home in a rented house in rural Cambridgeshire.

In the UK, use of KISS mode and TCP/IP packet delivery was quite widespread at that time, and operating in this mode was my first introduction to Linux-style commands. I used a system called NOS, and it offered far more sophisticated functions than regular AX25. The guy who operated our local node was a London tube train driver. I worked for an automotive consultancy in Cambridge, and they had Linux machines with an internet connection – so I was able to send real emails (using the low level SMTP protocol) from as early as 1991.

I was very fortunate to find a Maplin TU1000 terminal unit at one of the UK ham rallies, and was soon back on HF RTTY and AMTOR using the BMKMulty software. The TU-1000 was a well-designed unit with switched-capacitor filters and a digital tone generator, which used the same clock as the filters. I fairly quickly modified the TU1000 to add a ‘Hamcomm’ style interface (essentially an RS232 hand-shake line output with audio from a comparator), and that allowed me to also receive WEFAX and operate several digital modes using an upgraded version of BMKMulty.

Maplin TU1000

I later bought a used Kantronics UTU, which operated RTTY, CW and AMTOR, and threatened to make the Maplin TU1000 obsolete. However the user interface was essentially a dumb terminal application, and I found it rather clumsy to use compared with the excellent TU1000 and the convenience of BMKMulty.

I wrote my own terminal program for the UTU, which included all the common functions such as mode change programmed as Function keys. I still have the UTU, kept as a museum piece.

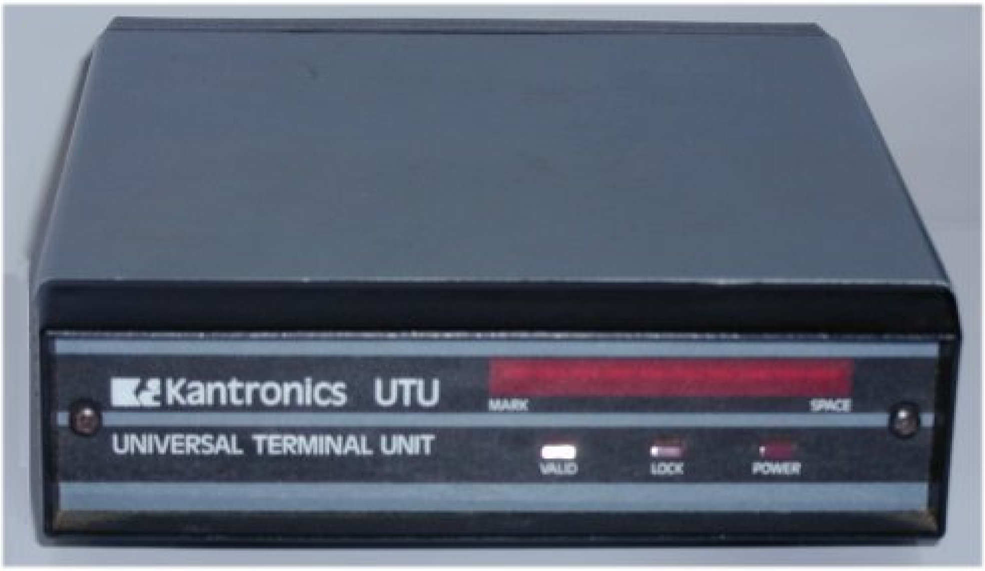


Fig. 7. The Kantronics UTU

3. Weather Satellites

I was first involved in receiving FAX around 1980. I had acquired some old mechanical FAX machines (a Xerox RX400, an Alden Marinefax and an old Muirhead full of valves with a scanner like a push-lawnmower reel, I remember).

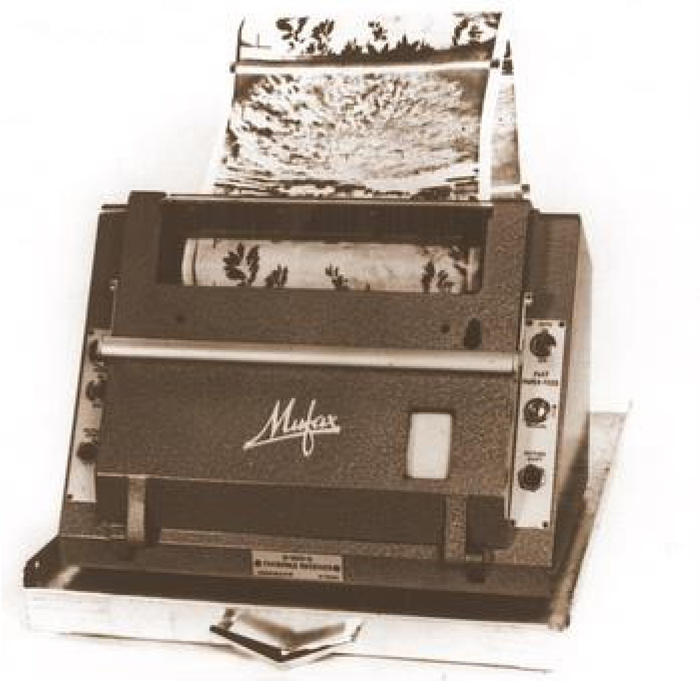


Fig. 8. The printing head of the Muirhead D611 FAX receiver (1950s)

These latter two machines used damp paper coated with starch and impregnated with Potassium Iodide. Current through the paper turned it dark brown. Resolution wasn't wonderful, and I was always short of paper. I ended up making my own!

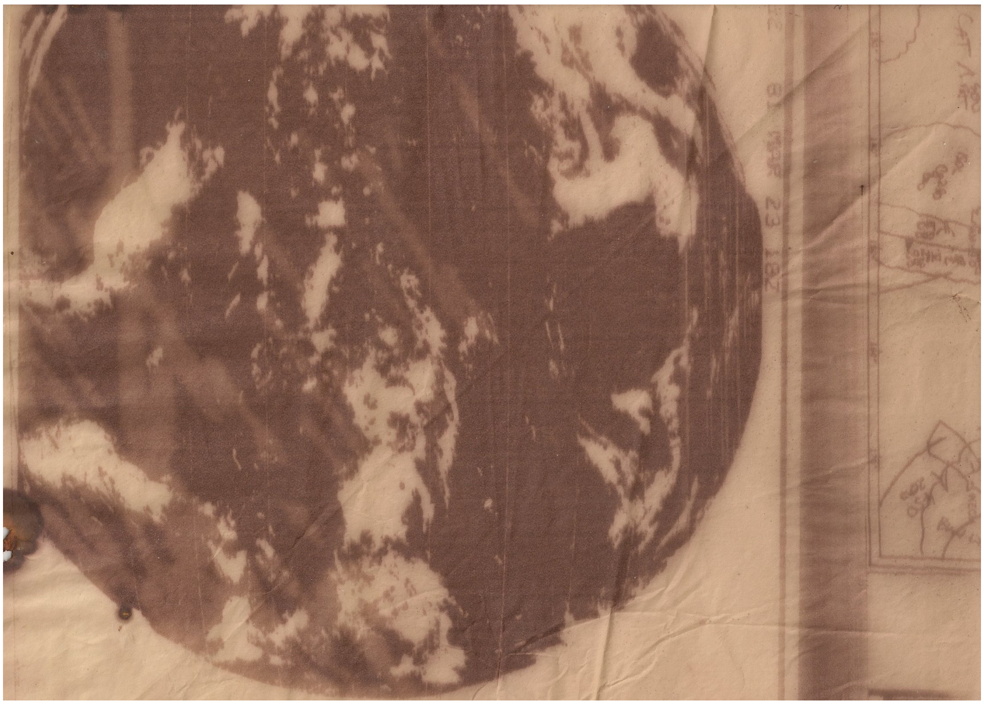


Fig. 9. Part of a wet-paper WEFAX picture, 1981

I recorded Figure 9 with the old Muirhead D611 machine from a NASA transmission on 11030 kHz, in March 1981. It shows a satellite view of the earth, but the resolution and contrast of these old machines was very poor. The results with weather maps (you can see part of one on the right) were somewhat better.

In about 1983 I started to put together a weather satellite reception system of my own. I built a receiver for 136 MHz out of an old AM taxi radio, and added a 10.7 MHz to 455 kHz FM conversion kit. The bandwidth was about right with the 10.7 MHz filter removed. Tuning was manual (and drifty), and when a satellite was due, I had to tune back and forth to find it.

By then I had acquired a much-improved facsimile machine, but it was a landline unit, which operated with FM signals. Since the satellite signals used a modulated 2400 Hz AM sub-carrier, conversion was required. This machine, made by Muirhead, had stepping motors, which allowed me to change speeds digitally, and the printing mechanism was a bit like a Xerox machine. It used china clay coated paper on which 0.5 mm scanning styli applied an electrostatic charge. This charge attracted toner particles from a bin, and these were then fused in place by a heater. I was able to use ordinary copier toner. The charge on the paper was generated from the signal by a high voltage pulse-width modulated supply, and promised quite good grey scale rendition.

To convert the satellite AM signals to FM was quite complex. I had to synchronously demodulate the AM and remodulate to FM using a voltage-controlled oscillator. To get the machine to start, and in phase with the picture, I needed to build a sync generator, some means to slip its phase, and a way to know when the phase was correct. The sync generator was used to control the VCO to start and synchronise the fax machine prior to switching to the incoming image.

In order to see where the image phase was, and to slip the sync generator appropriately, I used an old medical monitor with a slow phosphor. The horizontal scan was generated by a capacitor slowly charged partway to 100V and reset by my sync. Because the reset occurred at about 10 V, the charging and thus the scan was fairly linear. The vertical deflection of the monitor used demodulated video from the satellite, and it was quite easy to see where the image’s sync or picture edge was.

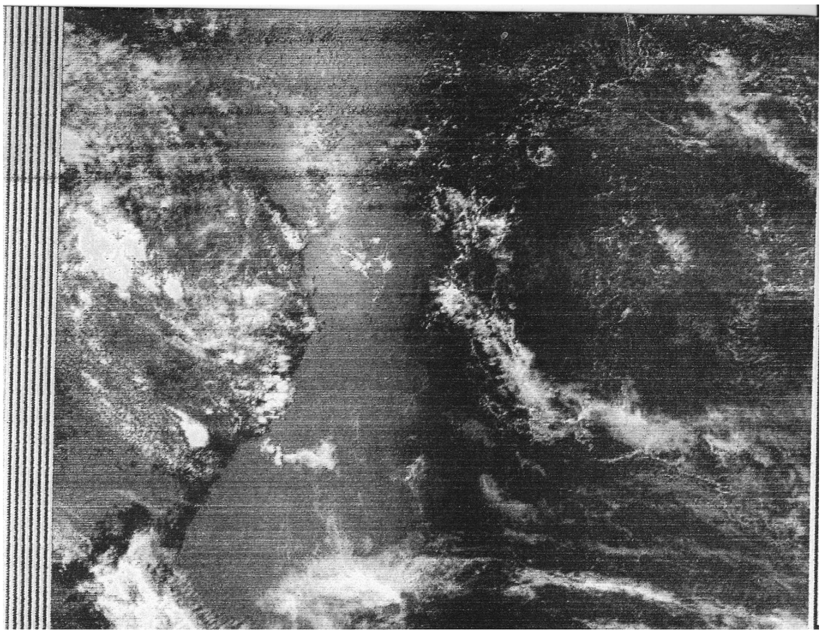


Fig. 10. Russian satellite MET15, showing the east coast of Australia, 1987

Figure 10 is the result. The machine gave quite good grey scale, and essentially noise-free pictures. This picture is from January 1987.

It took about five minutes to find the signal, start everything up and get the printer in phase, and so I built a beam antenna to point to the horizon, in order to 'see' enough signal from the satellite before it came over the horizon, so I could set the system running earlier. Once the satellite was over the horizon, I could switch to the crossed dipoles with circular polarisation, and achieve noise-free images from horizon to horizon. I was again always short of paper!

I later built a special A-D board which plugged into my computer AT bus, and was able to decode the satellite signals directly into the computer. Results weren't wonderful at the time, as I only had an EGA (16 colour, limited grey-scale) display. It's easy to forget how many limitations there were on technology back then!

In about 1991, now in the UK, I got hold of two bits of excellent software, WEFAX and SATFAX, which was commercial software.[3] It required special decoder modules, which plugged into and ran off the parallel port. I was able to borrow and reverse engineer these two modules. The software was excellent; perhaps a little limited in resolution due to pixel averaging, but as a result gave wonderful noise-free images.

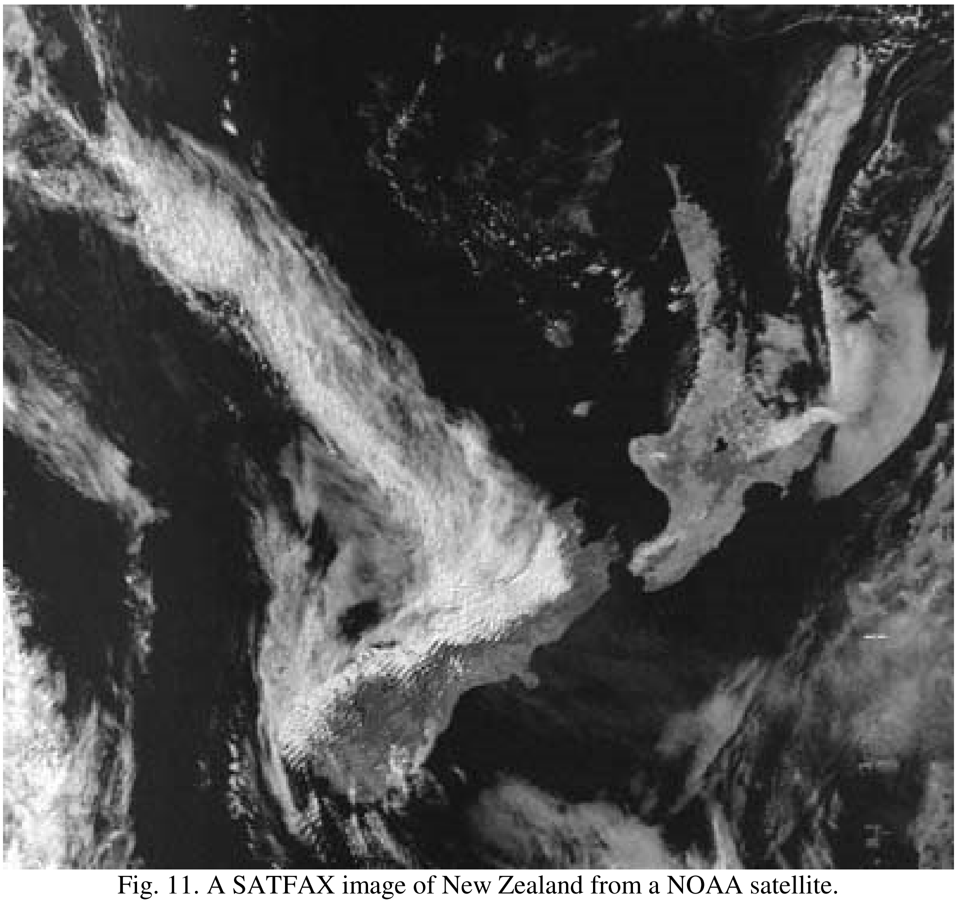


Fig. 11. A SATFAX image of New Zealand from a NOAA satellite.

In the UK I achieved many marvellous pictures with this software and a 386 computer with mono VGA display. I could see from Iceland down to the River Nile, and from the coast of Canada across to Russia. Figure 12 shows an example with Iceland, the UK and Scandinavia in winter, hiding under the cloud. Note the excellent grey-scale capability of this software.

[3] By Software Systems Consulting, 1990

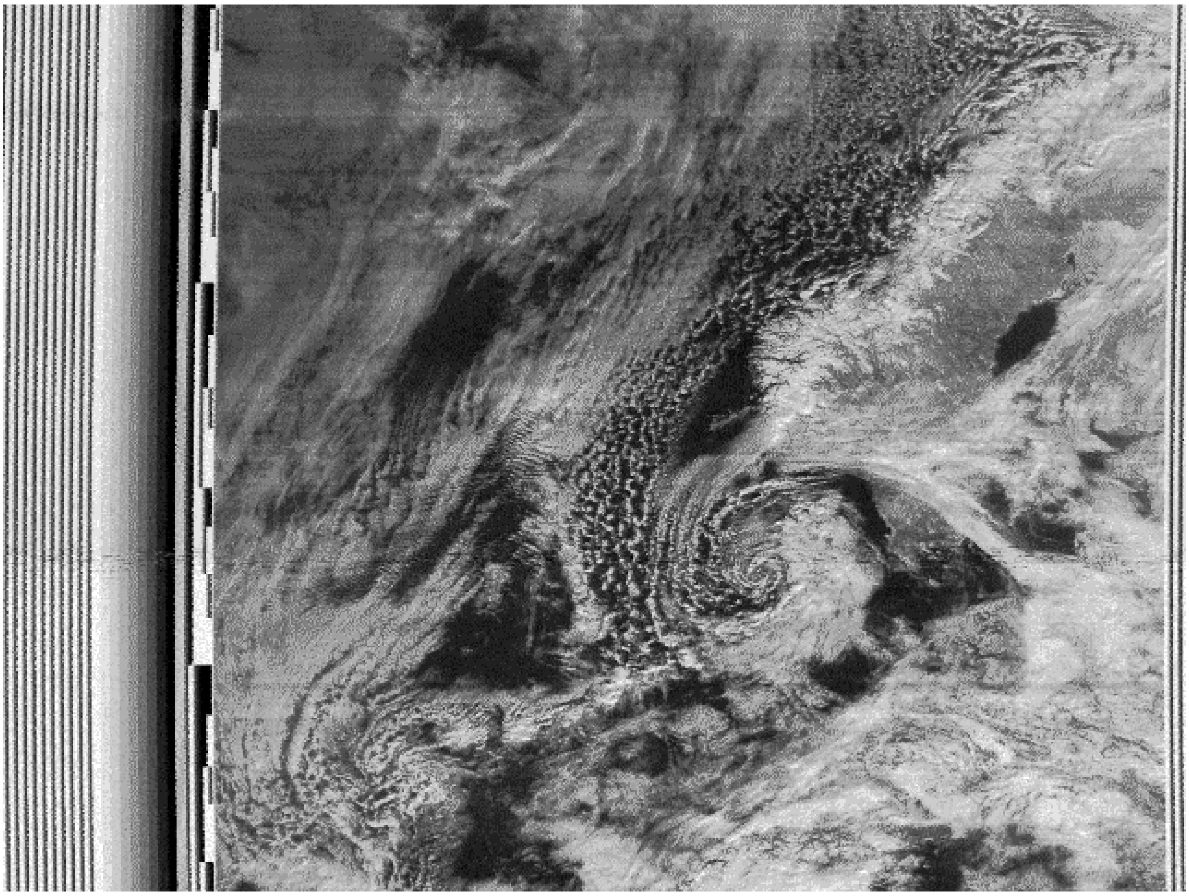
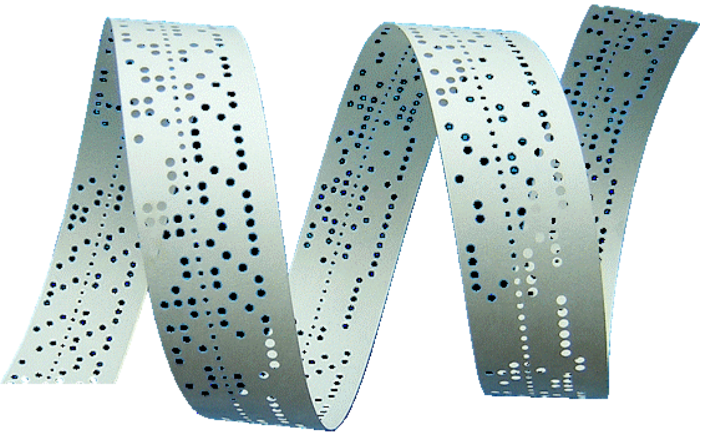


Fig. 12. Russian satellite MET-34, the UK, Iceland and Scandinavia

Later (home from the UK) I changed to using WXSat by Christian Bock, a sound card-based program written about 1998. This allowed images to be stored and manipulated, and I have many excellent archived pictures.



4. The 56002 EVM

Without doubt one of the most important steps towards improved amateur digital mode signal processing was a Motorola development kit, the DSP56002 EVM. The unit was popularised for amateur applications by Johan Forrer KC7WW[4]. Johan offered software for RTTY, and there were filter, noise reduction and spectral display programs as well. At the time, PCs were too slow to do serious signal processing, many lacked maths coprocessors, and sound card support was in its infancy.

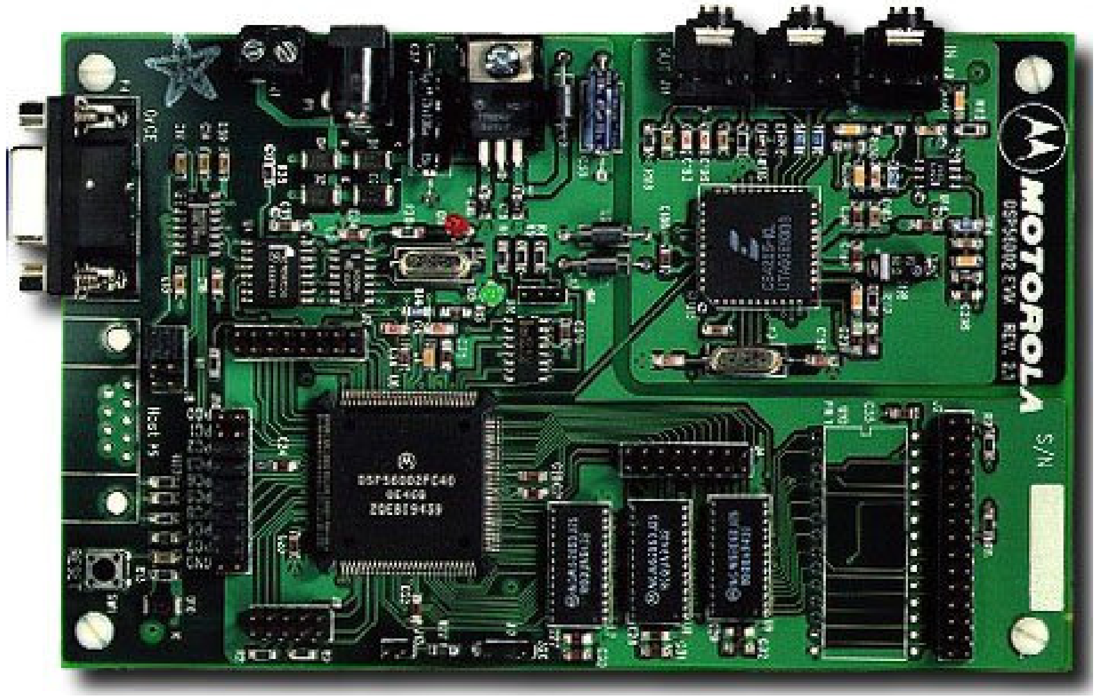


Fig. 13. The Motorola DSP56002 Evaluation Module

There were other similar products, from manufacturers such as Texas Instruments, and a variety of kits for amateurs from TAPR and other organizations. But the 56002 EVM was the most used platform for a few years. This relatively inexpensive board became popular around 1997, about the time I arrived home from the UK. It used a 40 MHz Motorola 56000 family processor with built-in hardware maths routines, which allowed complex maths to take place in a fraction of the time it would in PCs of the time. The board featured a 24-bit processor and high quality D-A and A-D.

It was not long before code appeared for 1200 bps and 9600 G3RUH packet modems. There was also software for weather satellite decoding and WEFAX (HF Fax).

It was at this time that the first of the uniquely amateur digital modes appeared – modes that would shape the future of digital mode development. An important feature of these modes was that they could include capability based on DSP that could not be practically achieved using hardware – specially shaped Finite Impulse Response (FIR) filters, Fast Fourier Transforms (FFT), quadrature modulation and demodulation, digital frequency synthesis and synchronous reception for example. This is a trend that has continued to this day, and we are now also seeing it appear in completely software defined receivers and transmitters.

[4] *Using the Motorola DSP56002EVM for Amateur Radio DSP*, Johan Forrer, QEX Magazine, August 1995.

Two names stand out from this era: Pawel Jalocha SP9VRC and Peter Martinez G3PLX. They developed modes for the EVM that survived the DSP era on into the sound card era.

Pawel was developing PSK ideas at the time, mostly for weak signal LF work. One of his bright ideas was MT63, a 64-tone parallel PSK modem with a rather complex Walsh-Hadamard Forward Error Correction (FEC) system that was remarkably robust. While it operated at only 10 baud, the default mode was 1 kHz wide, and since there was quite long interleaving of the coded data, it had an impressive ability to ride through fades. He wrote a terminal program for the DSP56002 EVM, and I used it to become the first person in the Southern Hemisphere on MT63. Les VK2DSG also bought a DSP56002, and we held the first QSOs in this mode outside Europe. Les features in my log many times in early 1999, along with a wide range of European stations.

Peter G3PLX is deservedly famous for his development of PSK31, but we need to remember that he single-handedly developed AMTOR (an amateur version of the marine protocol SITOR) in the late 1970s, and offered the microprocessor code (Z80 I think) for the mode to the world. It was to be incorporated into many of the multi-mode controllers.

Peter's first PSK31 code came about after studying the work by Pawel SP9VRC with his SLOWPSK mode for the EVM. Peter realised that slow differential PSK combined with a varicoded alphabet would provide an efficient narrow-band and reasonably sensitive DX mode. Again, the first code for this mode was written for the DSP56002, since the computers of the day did not have sufficient capability, and most had no sound cards. I was the first ZL on PSK31; I worked VK2DSG frequently, and also had the pleasure of working G3PLX on 20 metres.

PSK31 was intended as a chat mode for the HF bands, designed to be narrower than RTTY, and with better performance. It was first available in the mid-90s for EVM[5] and was released for Windows sound card in December 1998.[6]

[5] *PSK31: A new radio-teletype mode with a traditional philosophy*, Peter Martinez G3PLX http://www.qsl.net/dj9zl/psk31/p31g3plx.pdf

[6] RSGB Radio Communications January 1999. P31SBW108.zip

4. The 16-bit Sound Card

The first 16-bit sound cards, such as the Creative SoundBlaster, were available about 1992, using an ISA format, but at the time the computers available and the operating systems they used had rather limited sound support. In addition, there was little compatibility between the different manufacturers.

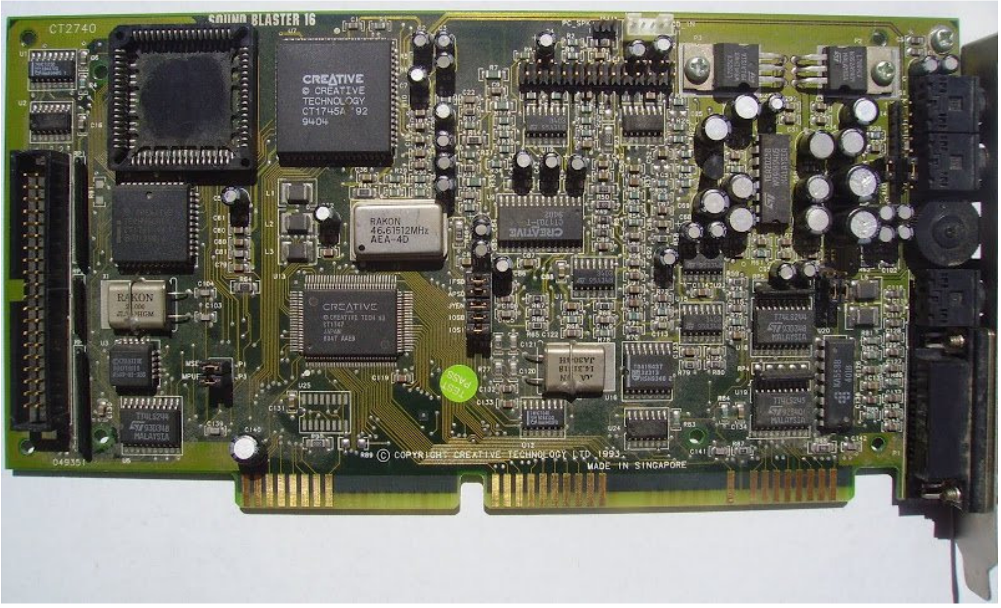


Fig. 14. An early Creative SB16 ISA sound card

Matters had improved somewhat by the time PSK31 for SoundBlaster appeared in December 1998. Windows 98 had sound card support, although still rather limited, but it meant that many more hams were able to operate PSK31, and the mode took on in a big way. 1999 was a time of excellent HF propagation, and my log shows PSK31 and MT63 contacts all over the world. MT63 had also by then been adapted to the PC sound card.

RTTY was also adapted to the PC sound card. One of the earliest programs I remember was RITTY, by Brian K6STI (1998). I also remember another program (for DOS) by Rob ZL2AKM. Soon after, there were numerous multi-mode programs offering PSK31 and RTTY, such as MMTTY and MMVARI, by Mako JE3HHT, who also offered the first really good (and free) SSTV software, MMSSTV, still among the best.

Although I operated a lot of PSK31 and RTTY at the time, I was never very happy with the performance. Far too often, RTTY signals, even strong ones, would only print rubbish, and PSK31 signals showed excessive phase shift, and would not decode. This led me to a period of extensive research, as I tried to understand what was happening to the signals. I found tools to measure the ionospheric effects, and picked the brain of an expert, Dr Gary Bold ZL1AN, at the time working at the Auckland University Radio Research Lab. It also led to my own first ventures into the development of digital modes.

At a later date (about 2008) I visited and had an interesting conversation with Dr Harry Whale, then retired. He (like me) was an Auckland Grammar old boy, and he studied at Trinity College in Cambridge. He was instrumental in setting up ionospheric research in New Zealand in the 1950s. He also imported New Zealand’s first computer, a differential analyser, now in the Auckland Museum of Transport and Technology. Harry died in 2009.

5. The Ionosphere

I don't intend this to be a lecture on radio propagation, as there is plenty of information out there already,[7] but over the course of the last 20 years I've become much better acquainted with what goes on 'up there', through research, by observation, and by making direct measurements. I've also put together some very good tools for examining propagation.

The problem with poor RTTY reception, I now know, is almost all to do with the signal time of arrival. When there are multiple paths, as there usually are on low HF at night, there can be up to 10 ms variation in the signal arrival time at the receiver. As one path fades and another takes over, the RTTY signal appears to move back and forth in time by as much as 10 ms, or two or more interfering signals can appear together.

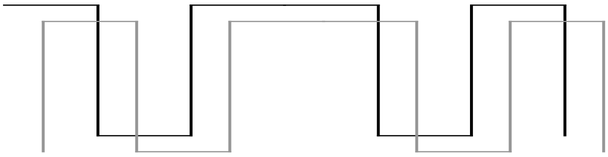


Fig. 15. The effect of multi-path timing and Doppler on an RTTY Signal

Since the signalling speed of 45 bps RTTY is 22 ms per symbol, an error of 10 ms can mean a gross distortion of the signal timing, and incorrectly selected data bits. This problem has nothing to do with the signal strength: this problem affects all NVIS and DX signals.

RTTY is also rather susceptible to selective fading, although this can be handled reasonably well by using narrow shifts and demodulators that are well designed, including DC restoration at the slicer (decision making) stage.

It's quite easy to measure these timing variations using Hellschreiber, a mode older than RTTY, but ideally suited to timing measurements since you can see the effects visually. Each column of a Hellschreiber character takes 57 ms to transmit. The transmissions are usually on off keyed, like Morse. If for example one station sends a row of dots, it would be very easy to see how much the timing varies over the radio path, and you can also see multiple paths.

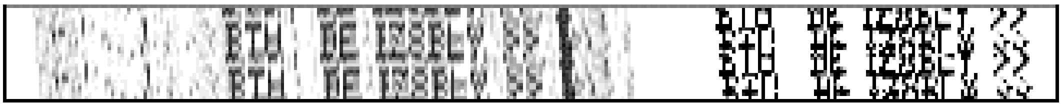


Fig. 16. Normal (left) and multi-path reception (right) from Italy on 20m

In this example, the left part shows normal Hell reception, with the text repeated above and below as normal. The right part shows rather stronger signals, but the text has arrived by short and long paths some 30 ms apart, so it appears like two stations at once.

The problems seen with PSK31 are rather more subtle, and have to do with ionospheric disturbance in Polar Regions. As the signal passes through the ionosphere over the poles, where there is a strong magnetic field and very mobile and spinning charged particles, the signals are phase modulated. The effect is easily as much as 90° on 30 metres, and can be twice that on 15 metres. This makes the signal essentially unrecoverable.

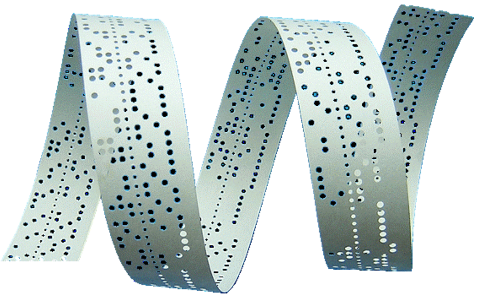
[7] *An Introduction to HF propagation and the Ionosphere*, M. Greenman ZL1BPU 2006 http://www.qsl.net/zl1bpu/IONO/iono101.htm

A further problem on longer paths, even those not over the poles, is that of scatter. The scattered components will have random phase (due to timing variation), and therefore the signal can appear to be periodically or continually phase modulated. So PSK31 is best suited to short-path DX, preferably single hop.

So having seen these effects and having an inkling as to the cause, I decided (a brave and naive move) that I could develop my own radio modes that would not be subject to these problems!

- To counter multi-path would require some mode with very slow signalling, or somehow not very sensitive to timing.
- To counter phase errors would require a mode that does not use slow phase-shift signalling, and yet could offer similar sensitivity to PSK31.

The modes I developed from here on all had this general concept in mind. How the problems were solved differed with the application (the modulation technique, and the bands or conditions the mode was designed for), but designing the best modes possible for specific applications was my aim.



6. Mosaic

Back in 1998 I had very little in the way of computer skills, and certainly no understanding of the way sound cards worked. At about this time, Peter G3PLX developed a parallel-tone experimental mode with 16 tones, (using the DSP46002 EVM) which would send text that could be read on a spectrogram. This wasn't entirely new, as a French company had demonstrated this capability before WW2[8].

By 1998 I had developed an interest in Hellschreiber,[9] and clearly this mode of Peter's was a new type of Hellschreiber, which came to be called 'concurrent-tone' Hell. The significant point that I grasped quite quickly from this idea was that the mode was not at all timing sensitive, nor particularly Doppler or drift sensitive. Each character took about 300 ms to send, and a variation of 10 ms was no problem, as all it would do is bunch up or spread out the letters a little. Frequency drift moved things up and down.

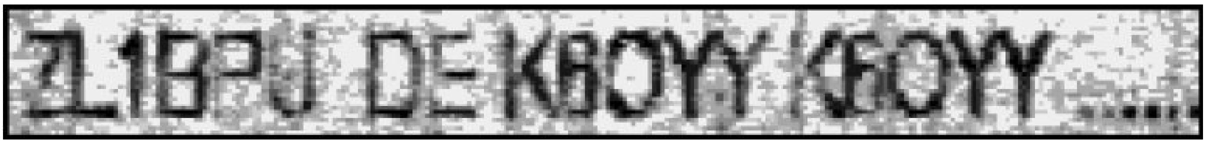


Fig. 17. K6OYY transmitting the G3PLX parallel tone mode

There was a similar system for DOS and sound card developed by Lionel Sear G3PPT around the same time. It used 9-pin dot matrix printer fonts and nine concurrent tones.

There were two problems with this mode. Because all the different tone frequencies (16 or 9 of them) could be transmitted at the same time, the peak to average ratio of the mode was rather extreme, which limited the average power, and put great demands for linearity on the transmitter. Because the average power was low, the mode lacked sensitivity.

But the whole idea set me thinking – could I develop a way to send characters, like Hellschreiber, but on different frequencies, one dot at a time? It would still have the relaxed timing requirements of the concurrent tone method, but would need only a CW transmitter with no need for linearity. I realised that I didn't need to know about sound card programming to develop a prototype, and that's how MOSAIC was born.

MOSAIC is essentially a 7-tone mode, seven dots, like Feld-Hell, although you can get away with five tones. I wrote the code in Quick BASIC, and used the PC speaker beep command to generate the sounds, which I coupled via a cable to the transmitter. The dot order was the same as for Hellschreiber, bottom to top, left to right, and I chose the speed and bandwidth initially to match reception using the G3PLX software, about 10 WPM. It also worked well received by early spectrogram software, 'Spectrogram' by Richard Horne.

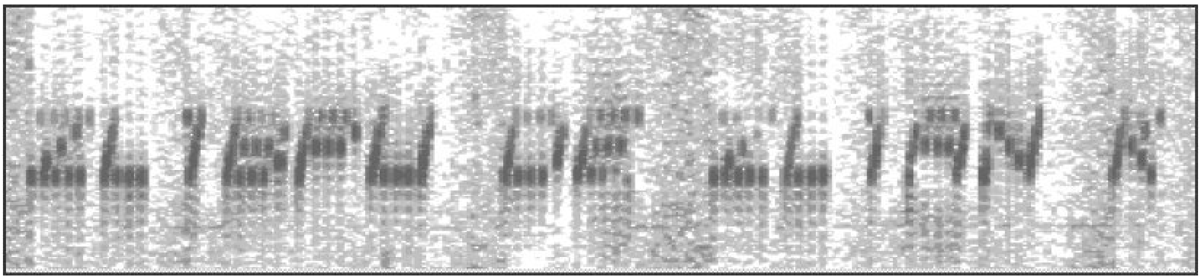


Fig. 18. First MOSAIC QSO, on 80m, 1999

[8] *A seven-frequency radio printer*, Electrical Communication, 1937, Le Matériel Téléphonique, Paris

[9] *Let's* ***See*** *you in Hell!*, Murray Greenman ZL1BPU, QST December 1999
http://www.qsl.net/zl1bpu/DOCS/CU%20in%20Hell.pdf

You can see in this early example that there is a grid of keying sidebands around the letters, and that's what led to the name MOSAIC. The text leans to the right because each column takes a finite time to transmit, and is sent one dot at a time from the bottom upwards. If you use only five dots per column, as in Figure 18, the text leans to a lesser extent than if more dots are used.

I put more development effort into the mode, which resulted in reduced and non-correlated keying sidebands, and allowed longer dot durations for a given text speed. This also improved the sensitivity. Several strategies were involved: (a) I used a variable width font to save time on narrower characters; (b) by reducing the duration of 'non transmitted' (white) dot spaces, the text could be sped up (at the expense of some character distortion); and (c) by sending each column twice at a higher rate, the text was clearer (like a diversity effect) and leaned to the right less.

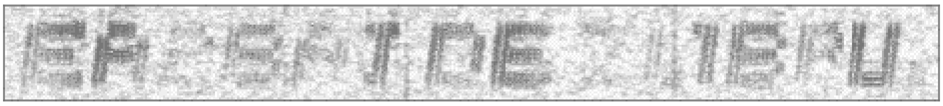


Fig. 19. Double column sequential-tone Hell (MOSAIC) received by EA2BAJ

Figure 19 is a record of the longest distance QSO ever achieved using multi-tone Hell. EA2BAJ was receiving me with the G3PLX software.

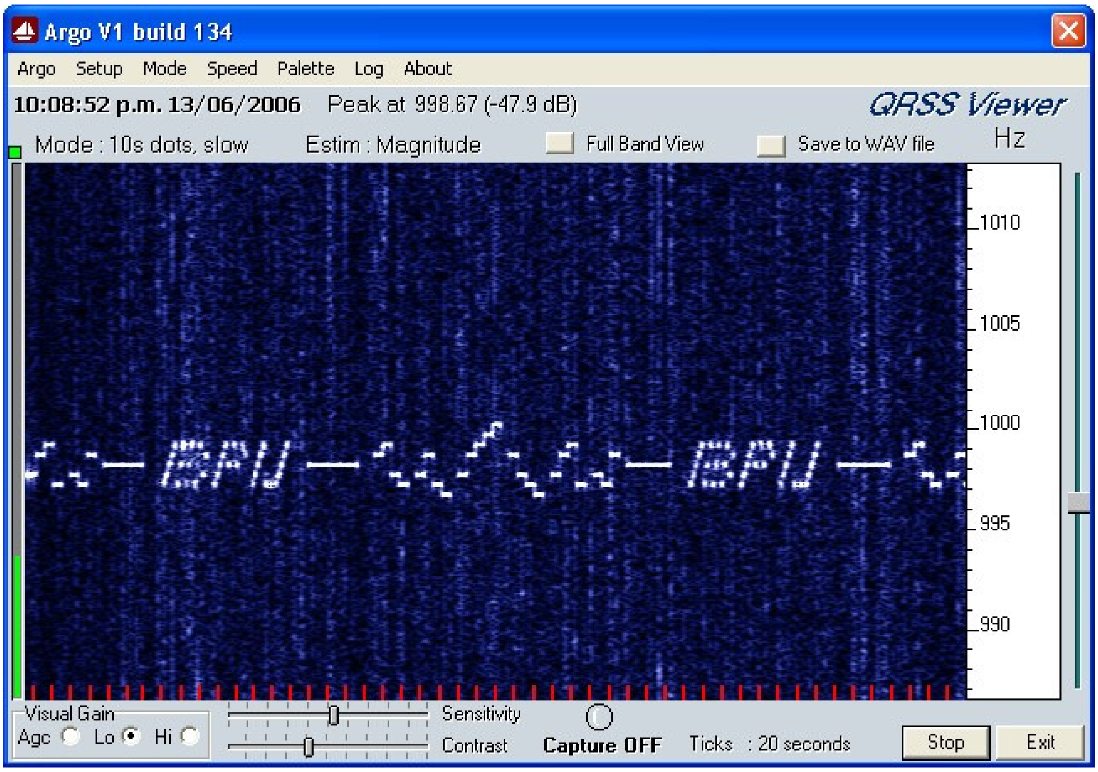
The MOSAIC software was the ***first*** application of sequential-tone multi-tone Hellschreiber anywhere, amateur or commercial. It is used very little in its original form these days, but is widely used in a slowed-down sound-card version for QRSS (very slow weak signal) applications. You don't need a linear transmitter to operate this mode.

One of the nicest of these QRSS applications is MEPT[10] Controller by Con ZL2AFP[11] (which I specified). It offers a WYSIWYG interface, where you click on a grid to make a pattern, which it can then transmit via the sound card, or save as a WAV file. It uses up to 16 dot positions, has a wide range of speeds and bandwidths, and can send different power levels in 6 dB steps. Just about any picture you can draw in 16 dots is possible.

One important advantage of sequential-tone MT-Hell is that it can be generated with a digital synthesiser. Another is that you can essentially send anything you like, including small graphics. And of course you don't need a linear amplifier. A CW transmitter will suffice, and this is an important factor on LF and MF, where transmitter efficiency is very important.

[10] Manned Experimental Propagation Transmitter (or transmission)

[11] *WYSIWYG MET Controller*, by Con ZL2AFP, http://www.qsl.net/zl1bpu/SOFT/WYSMEPT.htm



10:08:52 p.m. 13/06/2006 Peak at 998.67 (-47.9 dB)

Mode : 10s dots, slow Estim : Magnitude

Capture OFF Ticks : 20 seconds

Fig. 20. VK2DDI receives ZL1BPU on 472.5 kHz using ARGO.
Modes are Sequential MT-Hell and CASTLE

The example in Figure 20 shows reception by ARGO in 10-second mode. The transmitted signal was low power, (10 W, 100 mW EIRP) on 472.5 kHz, received by VK2DDI at a range of 2200 km.

The transmission source was a programmable DDS synthesiser (FEI FE-5650A) with bespoke control software, which I wrote. Each dot is of one-second duration, and the dot spacing is 1 Hz. There are five rows of dots per character.

The 'wiggly' mode in the centre of the picture is CASTLE mode – more about that later.

7. MFSK16

Some time in late 1998 I published a specification for a completely new mode, to be developed to match what I knew at the time about long-path DX propagation. On paths like this, PSK31 was not much good, RTTY was impossible, and although CW certainly worked on such paths, listening to the signals over the long path, they were frequently weak and 'watery': readable, but not clean.

By then I knew that long path DX had two principal problems – Doppler related phase and frequency errors, and multi-path timing errors. The short path could sometimes be viable, arriving 10's of milliseconds earlier than long path. Scatter was also a problem. Signals also tended to have significant fading, so I needed something to counter these problems. I didn't know as much as I do now, but my research (and experience with multi-tone Hell) led me to want to try an MFSK mode.

At that time, there were no amateur MFSK modes (apart from the multi-tone Hell modes), and few military ones, so this was all new territory. I had researched the two known diplomatic/military MFSK modes used at the time, Piccolo and Coquelet, and managed to find some useful papers describing the technical aspects and the claimed advantages.

It was clear that using multiple tones conferred improved sensitivity over RTTY. With more tone options per symbol, more data is transferred, and therefore a slower symbol rate did not necessarily mean a lower throughput. By using a slower symbol rate, the effects of multi-path were also potentially mitigated.

Piccolo

The British Foreign and Commonwealth Office developed Piccolo in about 1957, for diplomatic service.[12] It was first publicly demonstrated at an IEE exhibition in 1963. Perhaps the most influential paper describing the mode was that presented by J.D. Ralphs at this exhibition.[13]

This first Piccolo MFSK system operated at 10 baud, and used 33 tones, one for each letter of the ITA-2 alphabet (hold that thought!). The signal was about 400 Hz wide, implying a tone spacing of 10 Hz. The receiver output was converted to serial RTTY and operated a conventional 75 bps teleprinter. Piccolo used the same filter circuits to both generate the tones and detect the received tones. The receiver was synchronised to AM modulation at the transmitter for the first part of each symbol.

Piccolo was a clever system, but was outdated even by 1998, when I was thinking about this idea. The equipment occupied two racks and used dozens of valves. I needed something state of the art, but the description and theory of how the demodulator worked was very influential. It led me to understand the important relationship between signalling speed and tone spacing for optimum performance. No system before Piccolo had achieved such narrow tone spacing.

Unfortunately Piccolo used direct frequency shift keying, so required equipment with extremely high stability. The two expensive Plessey PR2250 receivers in Figure 21 have OCXO references.

[12] *Paper 204E*, H.K. Robin et al, Proc IEE, Vol 110, No. 9, Sept 1963.

[13] *Multi-tone signalling system employing quenched resonatorsfor use on noisy radio-teleprinter circuit*, J.D. Ralphs et al, Proc IEE Vol 110, No. 9, Sept 1963.

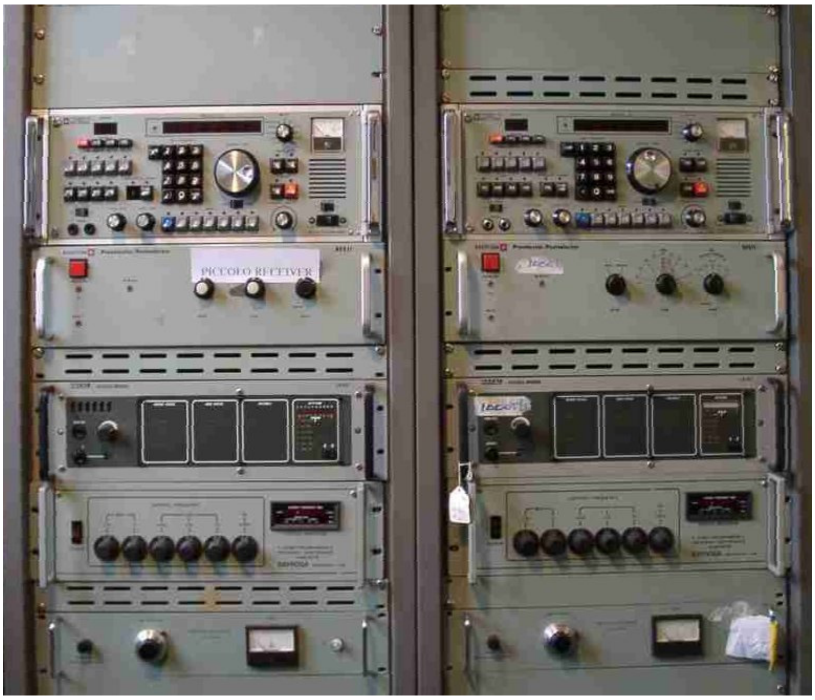


Fig. 21 Piccolo equipment now in the Bletchley Museum

Coquelet

Developed by ACEC in Belgium[14], Coquelet was a slightly older system, and was designed to be portable. An early version used 13 tuned reeds to act as filters, and also the same reeds to generate the tones. As a result the tones had a damped 'boing' effect as they sounded. There were two sets of filters, high and low, matched to the first two and the other three bits of an ITA-2 character. There were thus four high tones and eight low tones, plus an idle tone (the system was asynchronous, like RTTY). The earliest machines operated at 13.333 baud, equivalent to 100 bps RTTY. The machine was largely electromechanical, incorporated a teleprinter mechanism and keyboard, and was much simpler and cheaper than the Piccolo equipment. Each character was transmitted as a sequential high-low tone pair, and synchronisation was thereby implicit.

One interesting aspect of this machine was that on the back there were two rows of plugs, connecting the demodulator to the teleprinter mechanism, so you could muddle up the bit order, allowing for a simple encryption system with 32 possible arrangements.

MFSK16 Specified

In October 2009 I published a very comprehensive specification for what I confidently expected would be a very effective DX mode,[15] and which came to be called MFSK16 (it had 16 tones and generally worked at 15.625 baud). The mode was to run at 30 WPM. I invited comments and offers to program the mode for me, since I couldn't do it myself.

Much to my surprise, there was significant comment, mostly helpful. But significantly, there was also one offer to do the first coding effort. This was from young student Nino Porcino IZ8BLY, and within a matter of months his efforts resulted in the STREAM software.[16] The mode has since been incorporated in numerous other mainstream programs, including MMVARI, FLDIGI and MULTIPSK.

[14] Les téléimprimeurs, téléchiffreurs et transcodeurs ACEC – système Coquelet, ACEC Revue, No 3-4, 1970.

[15] http://www.qsl.net/zl1bpu/MFSK/tech.htm

[16] http://antoninoporcino.xoom.it/Stream/index.htm

There were numerous firsts incorporated in MFSK16, and there was understandably a lot of discussion about how the various aspects of the specification could be achieved. Nino was largely left to work out for himself how to achieve what was required, and this was all completely new territory. Briefly, the specification called for:

- A real-time chat QSO mode, with slick operation, 30 WPM.
- It was to include forward error correction with interleaver by default.
- It was to be easy to use, requiring no special equipment, and based on a 16-bit sound card.
- Transmission was to be continuous phase single-tone sequential 16-FSK.
- The bandwidth was to be 316 Hz for the 15.625 baud 16 tone mode, with tones spaced at 1/T (numerically the same as the symbol rate). This had never been done before using DSP.
- The mode was to be bit-stream oriented, allowing for FEC and the use of a Varicode.
- FEC coding was to be convolutional R=1/2 K=7, using NASA algorithms. This had also never been done before in amateur circles.
- Character coding was to be an extended ASCII Varicoded scheme. A grey code was to be used to weight the transmitted bits.
- The DSP receiver demodulator was to use an FFT filter as an integrate-and-dump replication of the Piccolo detector. This had never been done before.
- A waterfall tuning display was suggested (like that in PSK31) with point and click tuning.
- Decoded bits were to be weighed against all other FFT bins and assigned a soft decision status for the Viterbi-type decoder.[17]
- No sync was to be transmitted. Symbol sync was to be recovered from the FFT energy triangular characteristic (this had never been done before).
- Source code was to be made available, and the executables to be distributed free of charge to amateur users.

Nino and I also developed the first self-synchronising interleaver. Digital modes, and especially serial data stream modes, require synchronisation at a number of levels. In MFSK16 symbol sync was achieved through real-time examination of the FFT, which has a triangular characteristic. Interleaver sync was achieved using a fixed ‘square’ pattern of bit interleaving, where the weight of the bits decided the order. Since the weight of each bit is known from the 4-bit FFT decoder, synchronism was easy. Character synchronism from the bit stream was achieved using Costas symbol bits defined into the Varicode table, in much the same way as PSK31.

The first STREAM software was released before the end of 1999, and I published a related (award winning) article in QST soon after.[18] I had been running a web site[19] since about 1998, and the software was distributed via the web site.

In those early days, while tweaking and improving the software, Nino and I were in long-path QSO several times a week on 20 or 15 metres. One evening we had perfect copy both ways for two hours, until Nino’s transmitter overheated! Neither of us was using a beam antenna or high power.

STREAM also included BPSK31, QPSK31, MSK31 and MFSK8.[20] There were various ‘engineering’ options one could enable with the appropriate password, which allowed us to test various ideas, both useful and bizarre. After testing the options, we concluded that the

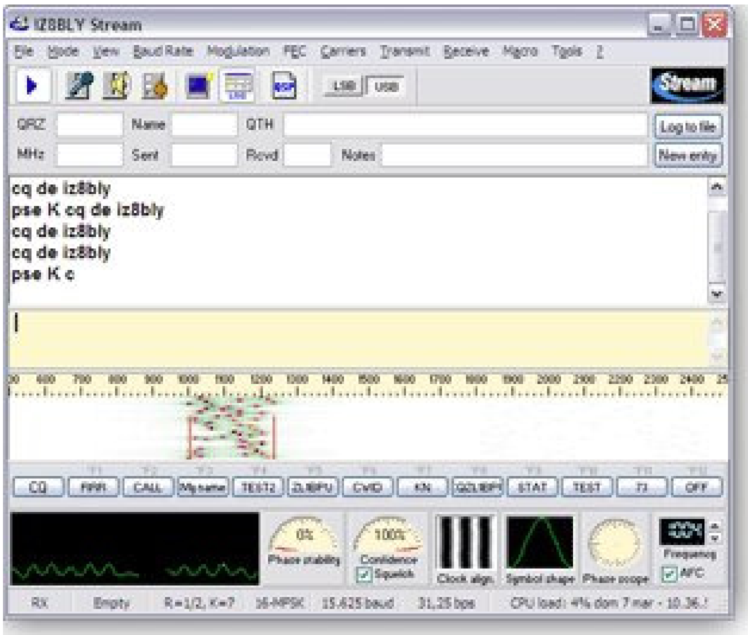
[17] This was suggested by Phil KA9Q, who was at the time the only guy with experience of Viterbi decoding (his boss at Qualcomm was Viterbi himself).

[18] *MFSK for the New Millenium*, M. Greenman ZL1BPU, QST Magazine, January 2001
http://www.arrl.org/files/file/Technology/tis/info/pdf/0101033.pdf

[19] *ZL1BPU Amateur Radio*, www.qsl.net/zl1bpu

[20] Same bandwidth as MFSK16, but used 32 tones at 7.8125 baud, about 19 WPM.

original specification was spot on. Improved versions were released until 2006, and by this time there were other implementations of MFSK16 available.



```
cq de iz8bly
pse K cq de iz8bly
cq de iz8bly
cq de iz8bly
pse K c
```

```
RX    Empty    R=1/2, K=7    16-MFSK    15.625 baud    31,25 bps    CPU load: 4% dom 7 mar - 10.36.
```

Fig. 22. An early version of STREAM

STREAM pioneered the use of ‘confidence’ meters (in this case based on Viterbi decoder metrics) and was the first application to use the triangular FFT summed energy signature to provide sync recognition (achieved by over-sampling the received symbols). It truly was a pioneering and groundbreaking program.

Good and Bad Points

There is no doubt that MFSK16 is a good mode, and certainly fulfilled the original aim of the specification. It became the mode of choice for long path DX, and remains so. Not everybody changed to it – even today you still see PSK31 and RTTY used in an attempt to work DX. Rather like taking a horse to water - you can’t make it drink!

The FEC worked wonderfully, and gave the mode impressive fade tolerance. We soon discovered a few disadvantages with MFSK16. At the time it needed a fairly fast computer (laughable these days), and it is not as slick as I would like – about 12 seconds turn around between overs. Of course if you employ an interleaver to provide fade tolerance, it comes with delays.

But the biggest issue (less so now with more stable rigs) was the fairly critical tuning and sensitivity to drift as a result of direct MFSK. From time to time there were also issues with poor reception when the signal seemed strong enough. It was eventually realised that MFSK16 did not have good tolerance of inter-symbol interference, i.e. a delayed or Doppler shifted version of a tone was liable to interfere with another tone, reducing reception quality.

This limitation was masked by the excellent performance of the FEC, and since that time the same problem has been noted with other designs. I tucked away the idea that the modem needed to be as good as possible *before* adding FEC, and would come back to it later.

All these limitations were to be addressed eventually in DominoEX and then FSQCall.

8. Hellschreiber

The new working relationship with Nino IZ8BLY led to further developments. During the MFSK16 development and after, Nino was working on software for the Hellschreiber mode.[21] He had at the time far more Hellschreiber signals to listen to than were available here. Around this time, Peter G3PLX had also just released a Hellschreiber program for the EVM.

Conventional Hell (dating from about 1925, and commonly called 'Feld-Hell') is sent as a series of pixels using a CW transmitter: key-down for black dots, key up for white 'dots'. The characters are sent as a dot matrix, scanned bottom-to-top and then left-to-right.

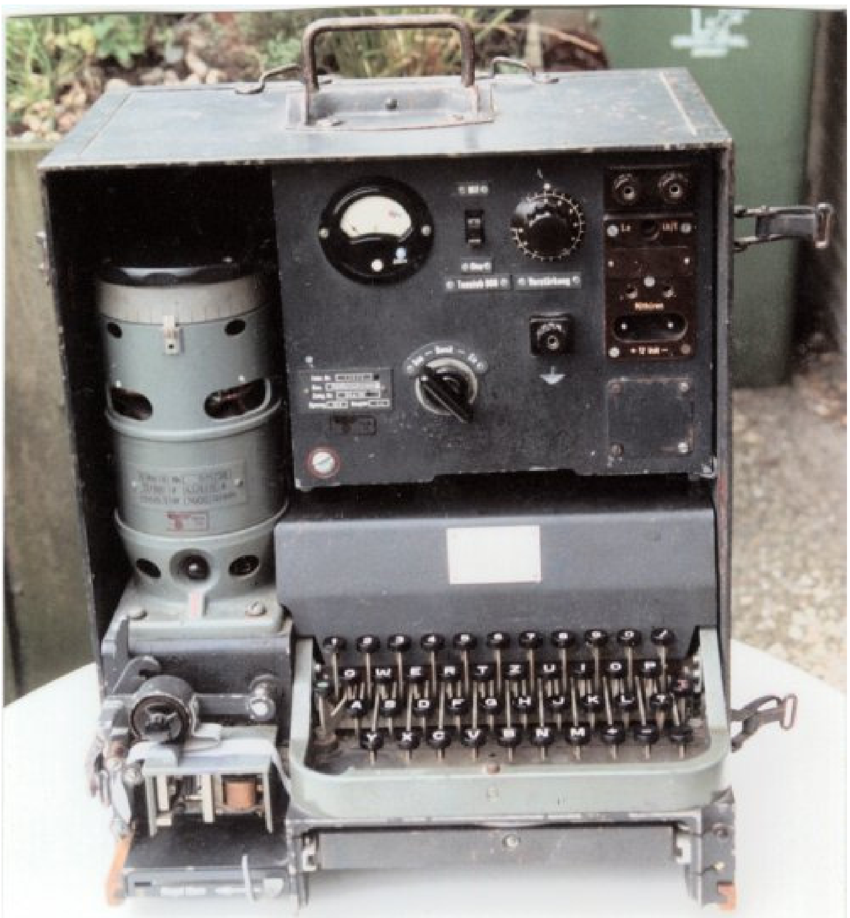


Fig. 23. Wartime Siemens A2 Hellschreiber machine (Dick Rollema PA0SE)

The mode traditionally operated at 122.5 baud, used a 980 Hz keyed tone (8 times the symbol rate) and had (in its simplest form) seven dots in each column. The Siemens A2 machine in Figure 23, and some other machines, had a clever way of doubling the text resolution without increasing the bandwidth. It was, like RTTY, designed primarily for landline use, but was used in the field by the Germans during WW2. Because of the relatively low duty cycle and the sensitivity of on-off keying to noise and fading, the mode wasn't suited to DX.

There had been Hell software available since the early days of home computers. One of the most popular for DOS was by LA0BX, which used a simple hardware modem and a parallel port interface.

[21] In fact Nino and I used his PSKHell as a fallback mode during the MFSK16 test phase.

I recall the first version of IZ8BLY Hell sound card software was in January 1999, and Nino and I had discussions about the limitations of the mode, which led to the development of PSK-Hell and FM-Hell. These modes used 100% duty cycle, and so reduced the noise and improved sensitivity, since phase detector demodulation is insensitive to amplitude variations. These modes were completely new, and there were numerous problems to overcome.

The worldwide first PSK-Hell mode QSO was between Gary ZL1AN and me in August 1999.

One of the subtlest problems was that of defining the phase of each character at the end of the character so it would not result in a phase discontinuity at the start of the next character. You can see this effect in Fig. 24, where small random black dots at the bottom of characters highlight the discrepancies.

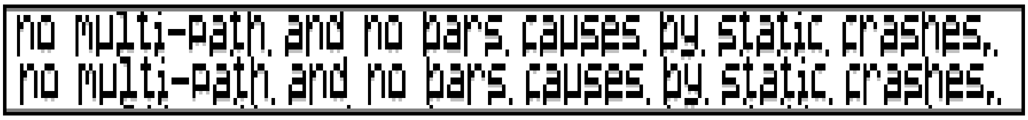
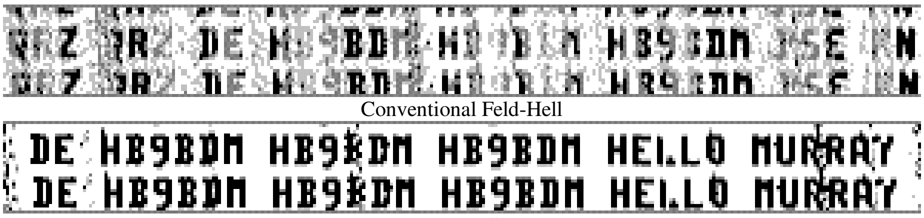

no multi-path, and no bars, causes, by, static, crashes,.
no multi-path, and no bars, causes, by, static, crashes,.

Fig. 24. Early example of PSK-Hell with phase discontinuities

In the end we defined a whole series of new fonts for PSK and FM-Hell to avoid this problem. FM-Hell is a subtle variation of PSK: actually it is MSK, so the phase change in each pixel is brought about by a slight frequency change, sufficient to change the phase by 180° over the course of each dot. A change in phase caused a black dot; no change in phase meant a white ‘dot’. The demodulator (phase detector) for MSK (FM-Hell) is identical to that for PSK-Hell.

The improvement offered by these modes was really marked. This can be illustrated (Figure 25) by two samples from the same QSO with HB9BDM, minutes apart – first in Feld-Hell, then in FM-Hell.


DE HB9BDM HB9BDM HB9BDM HELLO MURRAY
DE HB9BDM HB9BDM HB9BDM HELLO MURRAY

Conventional Feld-Hell

FM-Hell

Fig. 25. Feld-Hell (above), FM-Hell below.

We discovered through on-air tests and subsequent simulations, that PSK-Hell was occasionally prone to a strange ‘wave’ effect caused by multi-path, to which FM-Hell was not as susceptible (Figure. 26). The same multi-path caused only a slight thickening of the text in FM-Hell. Another major advantage of using MSK technology was that there was no need to use raised cosine shaping of the dots, and thus no tendency to generate unwanted sidebands if the transmitter was driven hard.

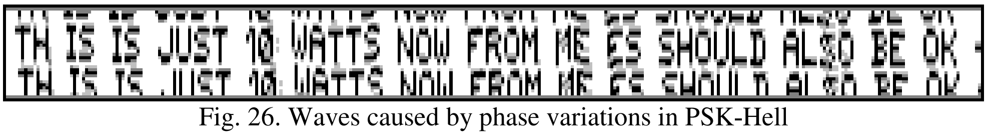

TH IS IS JUST 10 WATTS NOW FROM ME ES SHOULD ALSO BE OK
TH IS IS JUST 10 WATTS NOW FROM ME ES SHOULD ALSO BE OK

Fig. 26. Waves caused by phase variations in PSK-Hell

Nino and I developed two essentially semi-compatible variants, both operating at the same text rate as Feld-Hell. PSK-245 and FM-245 could use any Windows font, and although they were slightly wider, since they transmitted twice as many dots, there was no phase discontinuity, and they gave significantly improved image resolution and readability. The

PSK-105 and FM-105 variants used specially developed fonts, which were designed to maintain phase continuity. Of lower resolution, and lower symbol rate, they still resulted in the same text rate, but gave better DX readability and used less bandwidth.

The most recent version of IZ8BLY Hellschreiber[22] was released in 2004, and is still the best software for this mode. In addition to Feld-Hell, PSK and FM-Hell, this software operates a dual-tone FSK mode (Duplo Hell) that has symbols of twice the duration, and thus half the sensitivity to multi-path timing. It also has a concurrent-tone Hell mode and an excellent shaped dot transmit-only CW mode.

All the popular multi-mode programs now support Feld-Hell, and also have PSK-Hell and FM-Hell, just as we had developed them.

By the early 2000s, in the ham fraternity there was quite a bit of interest in micro-controller based 'beacons' of various types, and Hellschreiber because a popular mode yet again, because it was so simple to generate. I developed a keyer that would send (from a script) text in Morse, RTTY or Hell, and would also read analogue inputs and transmit the data as telemetry.

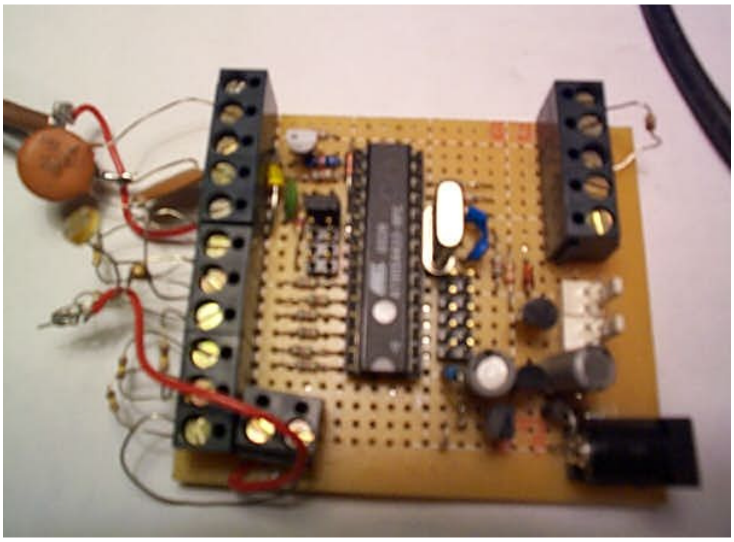
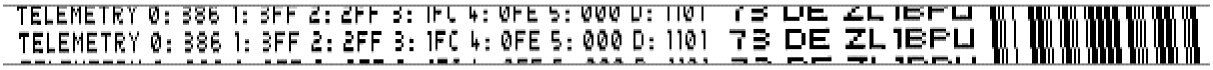


Fig. 27. The ZL1BPU Hell Keyer (used an AT90S4433 processor)

Many of the micro-based applications operated at slower speeds in order to achieve better range with low power: half-speed was common, achieved by simply sending each character column twice. This mode is illustrated in Figure 28. IZ8BLY Hell has half-speed (called DX mode), and also offers 1/8th speed and two fast (x5 and x9) speeds for Meteor scatter experiments.



```
TELEMETRY 0: 386 1: 3FF 2: 2FF 3: 1FC 4: 0FE 5: 000 D: 1101 73 DE ZL1BPU
```

Fig. 28. A telemetry frame sent by the ZL1BPU Keyer in Hell normal and DX modes

In Figure 28, DX mode text is on the right, followed by Morse, which you can just about read by eye, because it was sent one Morse element per Hell column, which worked out at about 11 WPM.

[22] *IZ8BLY Hellschreiber*, http://antoninoporcino.xoom.it/Hell/index.htm

9. MT63

I had operated MT63 as far back as 1998, using the DSP56002 EVM and software by Pawel SP9VRC. The mode was interesting, but the application, which operated like a modem with a dumb terminal, was a bit limited, and quite difficult to tune, since there was no visual tuning indication. There needed to be a decent Windows sound card version!

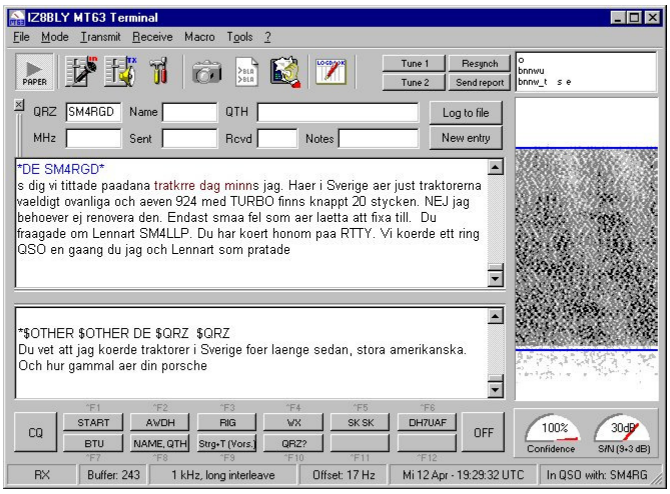
MT63 uses (in the default mode) 64 carriers spaced 15.625 Hz, and operates at 10 baud, 100 WPM. The carriers are 2-PSK modulated. There are narrower (5 baud, 500 Hz) and wider (20 baud, 2000 Hz) variants.

MT63 is something of a juggernaut, taking up a lot of band and making a lot of noise! MT63 uses highly redundant Walsh-Hadamard coding, effectively using a rate of 1/64. This made it robust, but also gave it the poorest bandwidth utilisation (10 Hz/WPM) of any mode since spark Morse!

It achieved an ill-deserved reputation for being sensitive, not borne out by measurements. It just *appears* sensitive because when the signal is not strong it is difficult to hear it among the background noise, since it sounds like noise, but will still decode. In reality, the sensitivity is no better than Morse (about –5 dB SNR). Tuning is a bit tricky, and although the typing speed is high (100 WPM), there are long delays (13 seconds with the long interleaver option) at both the start and end of each over.

The main advantage of MT63 is its ability to handle selective fading. It also handles in-band carrier interference very well. Designed by Pawel SP9VRC, the mode became accessible and reasonably easy to use once a Windows sound card version was available.

Nino IZ8BLY wrote the first Windows version I know of,[23] in November 1999. He ported a Linux version of the SP9VRC code, and added improvements, such as the waterfall, to make the mode easy to use. The final and definitive version was released in 2004. It is now also available in all the multi-mode applications, for example FLDIGI.



o
bnnwu
bnnw_t s e

QRZ SM4RGD Name QTH
MHz Sent Rcvd Notes

DE SM4RGD
s dig vi tittade paadana tratkrre dag minns jag. Haer i Sverige aer just traktorerna vaeldigt ovanliga och aeven 924 med TURBO finns knappt 20 stycken. NEJ jag behoever ej renovera den. Endast smaa fel som aer laetta att fixa till. Du fraagade om Lennart SM4LLP. Du har koert honom paa RTTY. Vi koerde ett ring QSO en gaang du jag och Lennart som pratade

*$OTHER $OTHER DE $QRZ $QRZ
Du vet att jag koerde traktorer i Sverige foer laenge sedan, stora amerikanska.
Och hur gammal aer din porsche

RX | Buffer: 243 | 1 kHz, long interleave | Offset: 17 Hz | Mi 12 Apr - 19:29:32 UTC | In QSO with: SM4RG

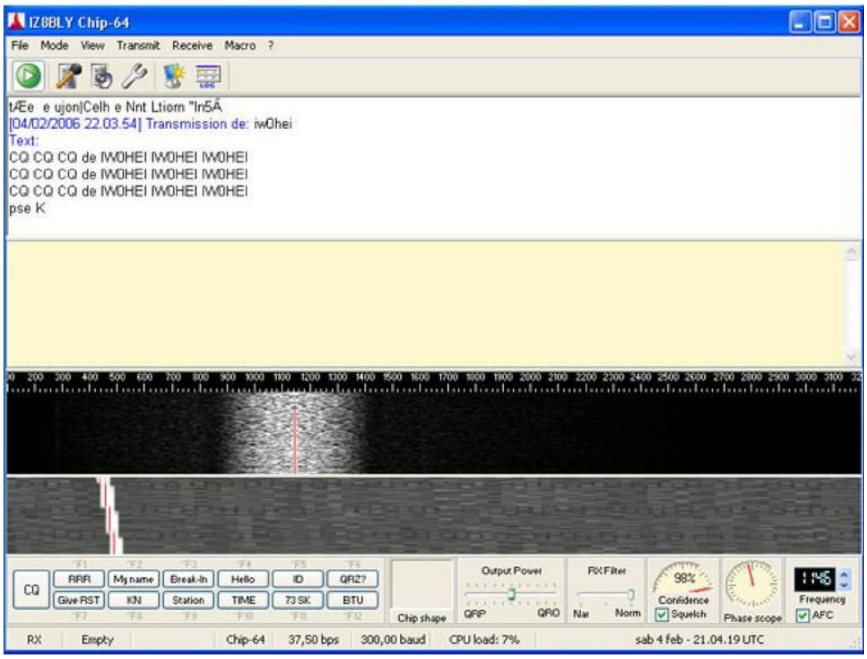
Fig. 29. The IZ8BLY MT63 user interface

The software had an interesting waterfall display which made tuning a lot easier, and also showed clearly any selective fades or interference within the signal bandwidth. The software also had secondary text during idle, ‘Confidence’ and ‘SNR’ meters.

[23] *IZ8BLY* MT63, http://antoninoporcino.xoom.it/MT63/index.htm

10. Chip-64

The next mode I became involved in was a radical departure, a spread-spectrum mode. The mode was not widely accepted, especially in the USA, where the symbol rate exceeded what was permitted below 30 MHz. It was tested first in Europe and between Nino IZ8BLY and I, and released in 2004.[24]



tÆe e ujon|Celh e Nnt Ltiom "In5Å
[04/02/2006 22.03.54] Transmission de: iw0hei
Text:
CQ CQ CQ de IW0HEI IW0HEI IW0HEI
CQ CQ CQ de IW0HEI IW0HEI IW0HEI
CQ CQ CQ de IW0HEI IW0HEI IW0HEI
pse K

Fig. 30. The Chip-64 user interface

The mode started out as an experiment, looking for ways to combat Doppler and multi-path problems that affected modes such as PSK31. So it was essentially a PSK mode, with a difference.

Chip-64 was probably the first amateur direct sequence spread-spectrum mode intended for HF use. It was coded by IZ8BLY, although I gave a fair bit of design advice, and I was involved in on-air and simulator testing. Chip-64 used a pseudo-random sequence multiplied (convolved) with the data bit stream, but at the radio frequency level was essentially just a fast PSK mode.

The most well known advantage of using spread spectrum is that on 'de-spreading' the signal at the receiver, the signal power is concentrated, and many of the interference effects of reception are cancelled or spread out, since they are not coherent with the signal. The reception process involves multiplying the data stream with the same pseudo-random (PN) sequence used to transmit it, but this needs to be in the correct phase relationship with the data (the multiplying bits must be in the same place).

This phase relationship is discovered using a *cross-correlator*, a very sensitive mathematical process that recognises a known mathematical sequence within another larger sequence. This was my first meeting with a cross-correlator, but was not to be the last!

By choosing an appropriate pseudo-random sequence (much research has gone into finding good sequences), the receiver can be made insensitive to random noise, and also surprisingly indifferent to transmissions using different sequences! In most cases spread-spectrum is used

[24] *IZ8BLY Chip-64*, http://antoninoporcino.xoom.it/Chip64/index.htm

to spread the signal over a wide bandwidth and so reduce the nuisance capability of narrow-band interferers. The GPS satellite signals are of this type, and each uses a different PN-sequence. Because Chip-64 had to fit within a 2.4 kHz SSB transmitter bandwidth, the aim was rather different, to give:

- Good tolerance of Doppler phase variations (by using a high symbol rate)
- To give good tolerance of timing shifts (by tracking the PN-sequence).

These are both problems with PSK31 that Chip-64 addressed.

Chip-64 offered two modes, both using 2-PSK at 300 chips/second. In Chip-64 the code length was 64 chips, in Chip-128 it was 128 chips. The characters to be transmitted were pre-coded into orthogonally unique sequences, and the resulting data rate was 37.5 bps for Chip-64, and 21 bps for Chip-128.

Chip-64 uses a Varicode based on the MFSK16 Varicode. The software had the first known example of a ‘Correloscope’, a real-time graph of chip phase versus time. When the PN-sequence is run across the incoming bit stream, one bit at a time, there will be little correlation until the correct phase is reached, at which time the correlation function peaks. It is this relationship that is illustrated by the Correloscope, which is a graph of time at chip scale (horizontal) and several seconds elapsed time (vertical).

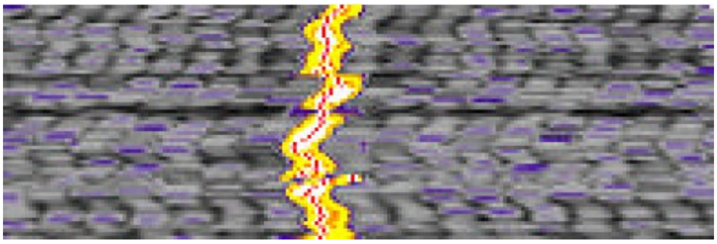


Fig. 31. The Chip-64 Corelloscope

Figure 31 shows the Correloscope output under CCIR Poor conditions, and it is clear that the correlation peak varies in time (the peak is the wobbly red line in the centre surrounded by yellow). The variation is ± about 5 chips, or about 20 ms, which the receiving software can easily compensate.

The correlator output not only steered the decoder, it also operated the receiver squelch to prevent print when the signal was lost. The sensitivity of cross-correlators is such that timing accuracy was not lost even on the weakest signals. In fact, you could detect the presence of, and tune in a weak signal by its cross-correlation, well before you could decode it.

Probably the main issue with Chip-64 is the need for very good sound card timing accuracy. Tuning could also be a little tricky, but no problem for a modern radio.

There was a lot of very sophisticated design in Chip-64. It included automatic frequency tuning, which was achieved by averaging and tracking the chip phase errors. We found that Chip-64 gave far better performance than PSK31 under multi-path conditions, although the weak-signal performance under quiet conditions was not so good, mostly because of the increased bandwidth used.

Chip-64 is not now used. In some countries there remain issues with spread spectrum use, even if no wider than SSB, and of course it has since been overtaken by better MFSK systems for long distance DX, such as DominoEX and THOR.

11. DominoEX

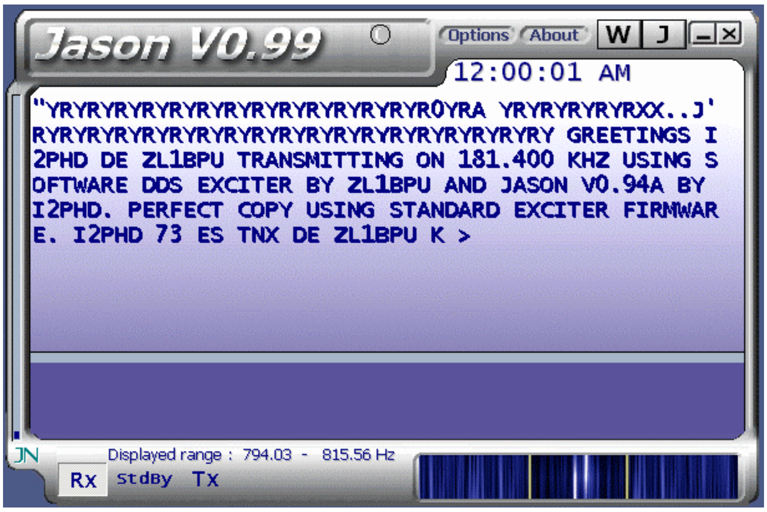
After the release of MFSK16, there was something of a hiatus of any development that involved me directly. At this time I was writing a book[25], and others were developing interesting modes. One of these was JASON, a weak signal mode for LF designed by Alberto I2PHD. Another was WSPR, developed by Joe Taylor K1JT. Poorly thought-out (and badly documented) modes such as Olivia and ROS also date from this time.

My own thinking about the limitations of MFSK had led me to consider a differential form of MFSK developed some years earlier (2001) by Steve Olney VK2ZTO (now VK2XV).[26] His mode, designed for a weak signal beacon, used what he called 'IFK', or Incremental Frequency Keying, where each symbol represents the sum of the previous symbol data and the new data. When received, the symbol numbers (from an FFT decoder) were subtracted to recover the data. Steve's intention in following this approach was to reduce the effect of drift in his simple transmitter, and this was achieved since provided the drift was modest, the subtraction process cancelled out the effect.

I had been thinking about the problems that MFSK16 had revealed, in particular drift and tuning intolerance, and also inter-symbol interference, and so stored away this IFK idea while also researching other approaches. At this time I also communicated these suggestions to both Alberto I2PHD and Joe K1JT, and would you believe, both ran with the idea of using IFK.

JASON

JASON is an IFK mode with 17 tones, sending ASCII characters in two sequential IFK coded symbols. The nibble (4-bit entity) order was recovered easily since the high-order nibble was always 8 – F (high bit set) and the low nibble always 0 – 7 (high bit cleared). With ASCII coding, this proved to be rather a slow mode, especially since it used only 0.08 baud in the default mode (about 0.4 WPM). But it was very narrow, although it had good sensitivity and relatively easy tuning for such a narrow mode (4 Hz bandwidth).



```
"YRYRYRYRYRYRYRYRYRYRYRYRYRYR0YRA YRYRYRYRYRXX..J'
RYRYRYRYRYRYRYRYRYRYRYRYRYRYRYRYRYRYRY GREETINGS I
2PHD DE ZL1BPU TRANSMITTING ON 181.400 KHZ USING S
OFTWARE DDS EXCITER BY ZL1BPU AND JASON V0.94A BY
I2PHD. PERFECT COPY USING STANDARD EXCITER FIRMWAR
E. I2PHD 73 ES TNX DE ZL1BPU K >
```

Displayed range : 794.03 - 815.56 Hz

Fig. 32. The JASON software

JASON also used (for the first time) an asynchronous synchronisation system (now called the 'peak hits' method), where a new symbol was discovered by simply noting when the

[25] *Digital Modes for All Occasions*, Murray Greenman ZL1BPU, RSGB, ISBN 1 872309 82 8
[26] http://joataman.net/narrowband/nb_comms/ifk.htm

maximum bin from the FFT detector changed. If at least three FFT solutions had been the same before the change, that was deemed to be a valid symbol.

Another interesting aspect of JASON was that it could (at my suggestion) output transmitter synthesiser commands via a serial port. In the above example, Figure 32, the message was transmitted in this manner via a ZL1BPU DDS LF Exciter.

WSPR

WSPR, developed by Joe Taylor K1JT, is also an IFK system, but operates rather faster (1.4648 baud) and uses only 4-FSK with a binary convolution FEC coding scheme. But it too uses IFK to minimise the effect of drift.

WSPR isn't a QSO mode, and employs a fixed frame of data and fixed (two minute) frames starting exactly on the UTC even minute. But it has excellent sensitivity, useful on LF/MF as well as HF (-29 dB SNR). It is deservedly popular as a worldwide propagation probe.

ISI Strategy

So having found a way to compensate for tuning difficulty and drift, I went back to the problem of how to avoid inter-symbol interference (ISI). At the time most of the research papers suggested quite questionable approaches, such as using strong FEC, using very wide frequency shifts, complex modulation schemes (seen in CLOVER, MT63, Olivia, ALE etc). I took the approach that if you designed the modulation correctly in the first place, none of these undesirable characteristics (too much bandwidth, too complex, too slow, limited advantage) would be necessary.

So I concentrated on finding an optimum modulation scheme for mid-HF DX. I was not too worried about bandwidth, nor about using multiple tones, as papers I had read suggested that up to 30 or more tones, MFSK systems kept getting better with respect to sensitivity.

While thinking about using IFK, I read some research papers that suggested convolving the data bit stream with a fixed or random pattern, in order to spread the tones and avoid ISI – something akin to an MFSK version of Chip-64. But this seemed unnecessarily complex; when all you needed to do was make sure that one symbol could not possibly interfere with the next.

A tone could only interfere with the next in an MFSK system if it used the same tone or an adjacent tone. Provided we used the optimum 1/T integrate and dump detector developed for Piccolo[27] and utilised in MFSK16, which should be easy! So, I proposed a modulation scheme that I called IFK+, where (in DominoEX), a fixed value of two was added to every differential modulating code. This was also a first for amateur chat modes at the time.

I had been somewhat influenced by the 'nibble coding' used in JASON, and so when it came to specifying a new mode (to become DominoEX), I suggested a nibble-based system, and designed a large nibble-based varicode alphabet for it.

DominoEX had a forerunner called Domino-F, about which the least said the better! It used IFK+, and was nibble-coded, but had two interleaved 16-tone sets (a primitive ISI strategy), and suffered badly due to errors in identifying the symbol order. But the basic idea was right, so we started again, and came up with DominoEX.

[27] Using the 1/T symbol rate/tone spacing relationship ensures that signals on adjacent channels are orthogonal, and are therefore optimally rejected. The symbol rate (baud) is thus the reciprocal of the tone spacing (Hz).

Enter DominoEX

I specified DominoEX to use a single 18-tone scheme with one, two, or three sequential symbols to define a character, a type of Varicode scheme.[28] For example, letter 'e' was defined as '1' (single symbol), while 'Z' was defined as '4, 15, 9', three symbols. Many of the lower-case letters needed just two symbols. The symbol order was easily identified since the initial symbol of each character was always in the range 0 – 7, and the subsequent ones in the range 8 – 15, thus signalling the character sync in the nibble most significant bit.

Tone set management for minimum ISI was achieved by simply shifting the intended transmitted tone up two steps at each symbol. This ensured that the next symbol, even if the same character, could not be on the same or an adjacent frequency, and thus could not cause ISI. By using minimum (1/T) spacing, any interference caused by spillover from an adjacent FFT bin was also eliminated.

Notice that in this development, no mention has been made of FEC. I was trying to design an HF chat mode, and FEC would be a last resort, since it adds latency, slows the transmission rate, typically by a factor of two, and generally destroys the pleasure which can be obtained by 'slick' operating modes designed for chat – PSK31 being a good example. I wanted to make the modem so good at a fundamental level, that FEC would not be required.

It transpired in later tests that DominoEX at 8 baud (50 WPM and no FEC) gave better reception, lower delays, and higher text throughput than DominoEX with FEC (or THOR) at 11 baud (35 WPM)![29] This same effect led us to exclude FEC from the development of FSQCall (see later).

Con ZL2AFP Enters the Picture

An important factor in this development was discovering a local ham (well, Wellington, 400 km away) who was a keen digital modes user, and who had quite good programming skills. More significantly, he has good advanced maths skills. I met him first over the air, and we discussed current modes and future developments for some time before I suggested in 2004 that he have a go at what was to become DominoEX.

The program name, by the way, means 'Domino, Extended'. 'Domino', because the waterfall in the early programs looked rather like the dots on a domino, and 'extended' because it used an extended ASCII character set with 256 characters in two complete sets. This was the first time more than 16 tones had been used for an Amateur digital mode.

As far as I can recall, the first development we did together was DominoF, then DominoEX, starting in 2004. The programs were written in PowerBASIC. The final, definitive version of DominoEX was released in 2006.[30] It included (as an option) convolutional binary FEC, achieved by changing the character set to a binary one rather than a nibble-based one.[31] The convolutional coding employed was the same NASA standard R=1/2 K=7 as in MFSK16.

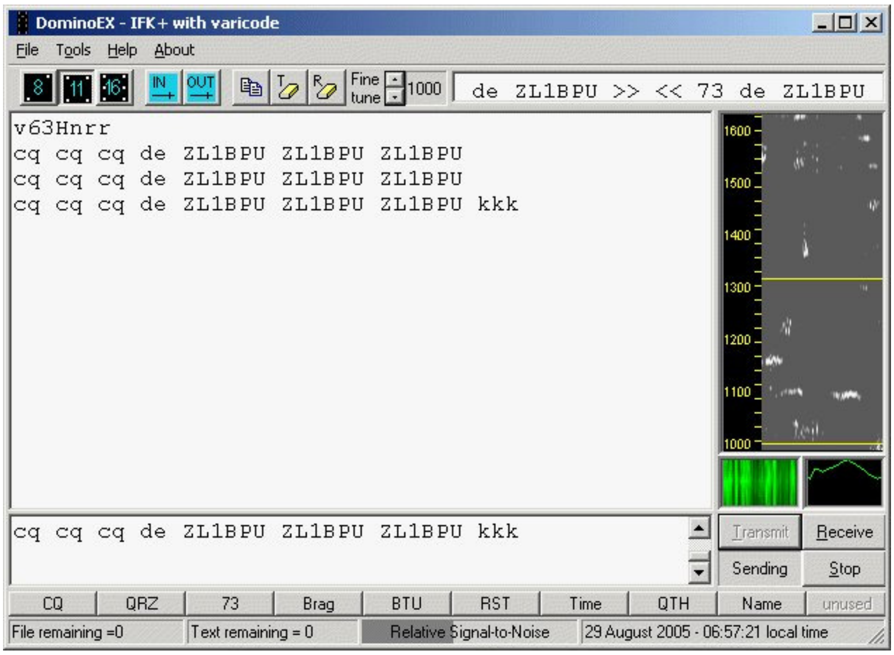
By 2006, DominoEX had also been incorporated into FLDIGI by Dave W1HKJ and MultiPSK by Patrick F6CTE. An improved FEC version called THOR appeared in FLDIGI soon after.

[28] http://www.qsl.net/zl1bpu/MFSK/DominoEX%20Varicode%20Table.pdf

[29] So the coding gain achieved at half the symbol rate exceeded that of using a rate ½ FEC system!

[30] www.qsl.net/zl1bpu/MFSK/DEX.htm, www.qsl.net/zl1bpu/SOFT/DominoEXFEC%202.0d.zip

[31] The same character set as MFSK16, with secondary text extensions.



```
de ZL1BPU >> << 73 de ZL1BPU
```

```
v63Hnrr
cq cq cq de ZL1BPU ZL1BPU ZL1BPU
cq cq cq de ZL1BPU ZL1BPU ZL1BPU
cq cq cq de ZL1BPU ZL1BPU ZL1BPU kkk
```

```
cq cq cq de ZL1BPU ZL1BPU ZL1BPU kkk
```

File remaining =0 | Text remaining = 0 | Relative Signal-to-Noise | 29 August 2005 - 06:57:21 local time

Fig. 33. The ZL2AFP DominoEX user interface, an early version

DominoEX later introduced further 'metrics' for the user to watch and marvel at: the FFT energy sync triangle, a sync history, Doppler history, signal strength and FEC confidence.

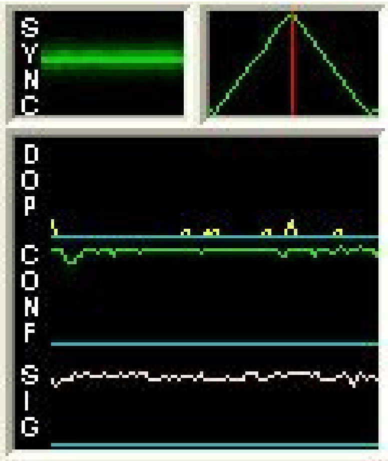


Fig. 34. The DominoEX metrics

Like MFSK16, DominoEX is a synchronous mode, again using the triangular FFT energy characteristic to discover sync. Because there are never any repeated symbols (thanks to IFK+), synchronism is secure, even with very weak signals.

Rather than use an idle character to maintain sync while temporarily out of text to transmit (as in PSK31 or MFSK16), DominoEX had a complete 'secondary text' alphabet. While all these secondary characters used three symbols, and so weren't that fast to send, the whole point of using them was to fill in time. The user could specify a text string to be sent during idle, and this was shown at the receiver in a separate little 'Times Square' scrolling display (at the top in Figure 33).

The Secondary Text idling idea was first suggested by me in 1999, and first incorporated in the IZ8BLY MT63 software. It was later used in CMSK as well.

DominoEX certainly fulfilled the requirements I had in mind. With a slightly slower symbol rate than MFSK16 (11 baud), it had good sensitivity, no problems with drift and tuning, and quite good long path and multi-path performance, thanks to the use of slow MFSK. The performance on weak and fading paths wasn't as good as MFSK16, but switching on the optional FEC, or using the THOR version, quickly corrected that.

FLDIGI THOR

At about the time DominoEX was first released (2004) I was in discussion with Dave Freese about the addition of FEC to DominoEX, and as I mentioned, he made his own version, which is clearly much better. The principle is similar, and he used some of my suggestions, and even more of his own, to come up with a much improved receiver. He used symbol-wise soft decision metrics from the FFT decoder (strength of the biggest peak relative to the mean level, and a Doppler metric) to inform the soft-decision Viterbi decoder. In this he also included code puncturing based on the total energy in the FFT at each decision, which was a good indicator of lightning static.

Dave called this mode THOR (lightning connotations), and it has proved to be the best mode for the US 'lightning alley' (south-east coastal states) where there are times when no other mode makes any headway against the QRN. Performance is impressive. The only problem with THOR (this also applies to the DominoEXFEC version) is that the typing speed is halved and there is significant latency between overs. But the result is perfect print!

DominoEX proved to be a reliable and easy to use mode for DX on 20, 30 and 40 metres. Without FEC it lacked the fade tolerance of MFSK16, but the ability to handle Doppler was a huge benefit, and it is a lot easier to use – easy to tune and no drift problems. Without FEC it is also very 'slick', no delays between overs.

DominoEX is also effective on 80 metres. Until it came along, the alternatives for 80m (RTTY, PSK31) did not perform well, due to high symbol rates and poor multi-path performance. AMTOR A operates at 100 baud (it still isn't available for the sound card). I have tested it, and it's very slow, with multiple retries. I have also tested AMTOR B on 80m, and it's not wonderful either.

One interesting aspect that was discovered on the US East Coast was how well DominoEX worked on VHF/UHF FM. It was discovered that you could copy signals too weak to open the FM receiver squelch, and also that the sensitivity was better than VHF SSB or FM by a wide margin. The same has been proved true more recently for FSQCall. Of course using FM with an audio subcarrier on VHF/UHF also avoids problems with drift, and makes operation accessible to hams without VHF/UHF SSB transceivers.

The lifetime of DominoEX has been quite long. Just this year (2018) Dave W1HKJ has released a much narrower version, DominoEX Micro, intended for use on LF/MF bands where in some countries the permitted signal bandwidth is quite restricted. The bandwidth is about 36 Hz, and there is a THOR Micro version as well. These modes operate at 2 baud. THOR Micro is of course quite slow (3 WPM), but has impressive sensitivity (-18 dB SNR).

Another advantage of the FLDIGI Micro versions for ZL operators is that it allows us to use a sensible digital mode on 60m. An arbitrary 100 Hz bandwidth rule has excluded FSQCall, the optimum mode for this band.

12. Eclectic Developments

Once DominoEX was done and dusted, and no further major developments were in mind, Con and I went through a period of cooperation developing things for our own pleasure. Con has long been interested in image transmission, especially narrow-band TV (NBTV), so this was the first mode to receive his attention.

A series of useful (and some bizarre) modes evolved from 2007. Most were rather like sped-up SSTV, although they used a sequential frame arrangement, rather than the single-frame SSTV business. SSTV had largely lost its sequential frame capability way back in the 1970s, and was securely single-frame when the Robot 1200 frame-store equipment came along in 1984.

Analogue NBTV

Probably the most successful of these new modes was the Analogue OFDM NBTV system,[32] of which there were five modes, both B&W and colour, with frame rates ranging from one per second to one every six seconds. The receiving software could save the sequential frames as a movie, so they could be played back at a speed that allowed for movement to be visualised.

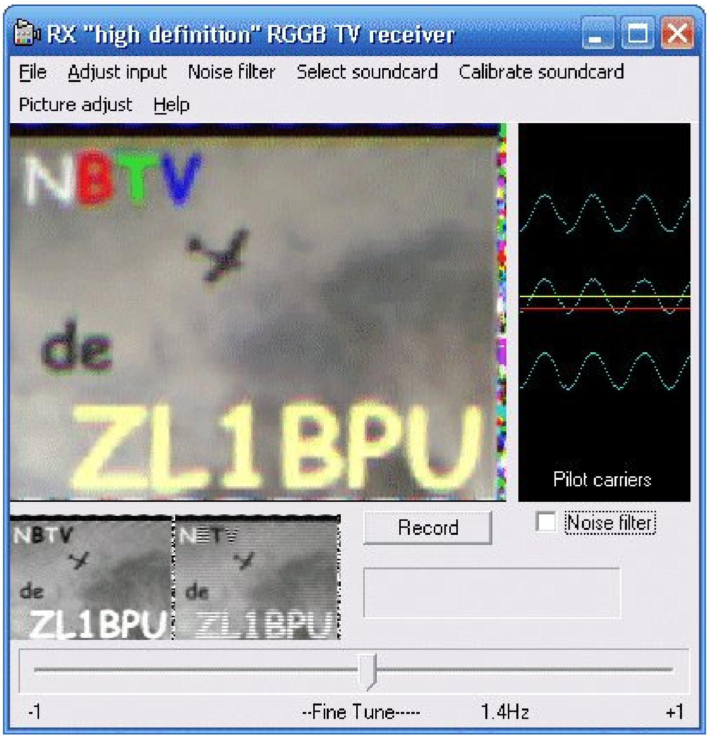


Fig. 35. The RGGB receiving software

The OFDM system was based on a 2 kHz wide parallel tone modem, with very close-spaced tones. The spacing was 42 Hz or 32 Hz, depending on the mode, and the carriers were individually FM modulated just a few Hz, each with one line of the image. The pixel rate was correspondingly 50 Hz or 33 Hz. The fastest version was able to send a 48 x 48 pixel B&W image every second, so although the resolution was limited, it was made up for by the eye processing repeated frames. The purpose of this approach was to use a very slow line rate, all lines sent concurrently, which would overcome the limitations of SSTV type modes on the low HF bands, where timing variations and fading caused endless trouble. The 48 x 48 pixel modes used a 1:1 aspect ratio format, the others a 4:3 format. The pictures were scanned from

[32] http://www.qsl.net/zl1bpu/NBTV/OFDM.htm

left to right. To send a portrait mode picture, you were required to rotate it 90° before and back after transmission.

Colour was achieved by one of two methods: sequential RGB frames (so colour was three times slower than B&W – see Figure 36), or an RGGB method, where colour differences were transmitted (Figure 35). The latter version was especially successful on VHF and ground wave paths, but was rather sensitive to multi-path effects, which distorted the colours.

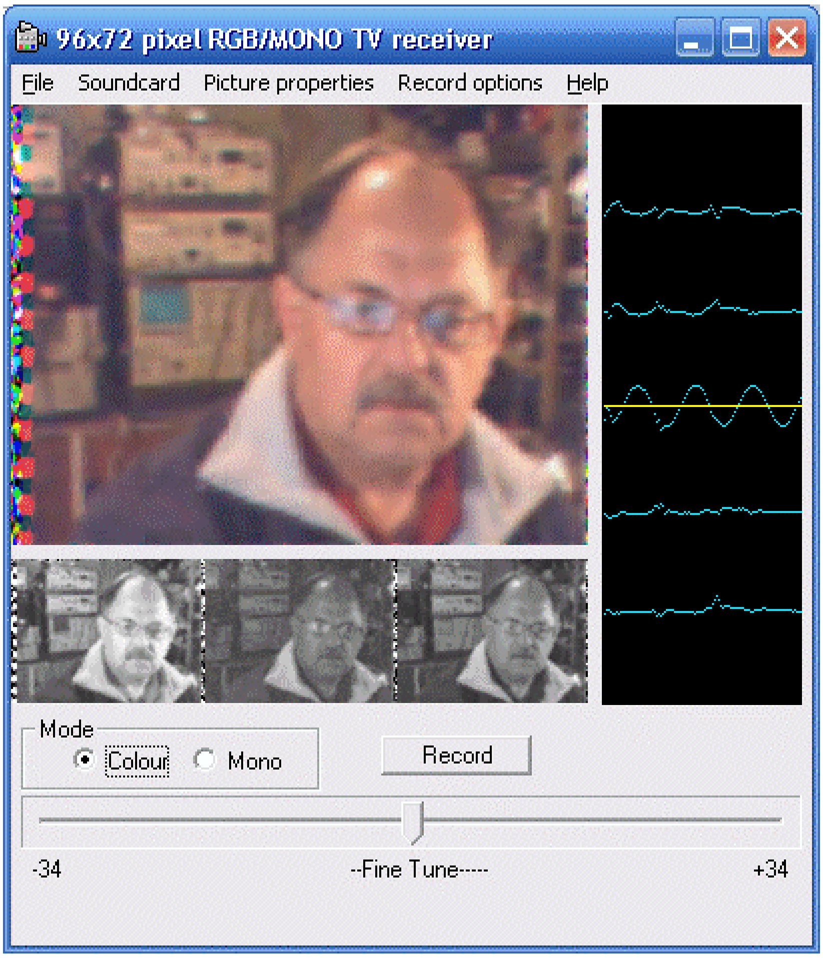


Fig. 36. A shot of the author in 96 x 72 pixel RGB mode

The modes were not sensitive to multi-path fading, apart from the picture noising up in the deepest fades. This could be effectively countered using a sequential frame averaging option in the playback software.

In Figure 36 you can see the enlarged (interpolated) received colour picture, to the right the pilot carrier tuning display, and below the native size raw frames, red, green and blue. Synchronism to the frame sequence was achieved manually, by clicking the mouse to the left of the first frame in the sequence. Some experimentation was required.

In all these modes, tuning could be extremely difficult, so at my suggestion Con introduced a pilot carrier in the middle of the picture, distinctive because it was modulated with a slow sine wave. The pilot carrier-tuning indicator (on the right in Figure 36) was a graph of frequency vs. time (from the centre of the FFT demodulator). Only five carriers were displayed, and the idea was to centre the pilot carrier as the centre of the display, either by tuning the receiver (very difficult to get closer than 10 Hz), or with the Fine Tune adjustment, which had a range of ± 34Hz.

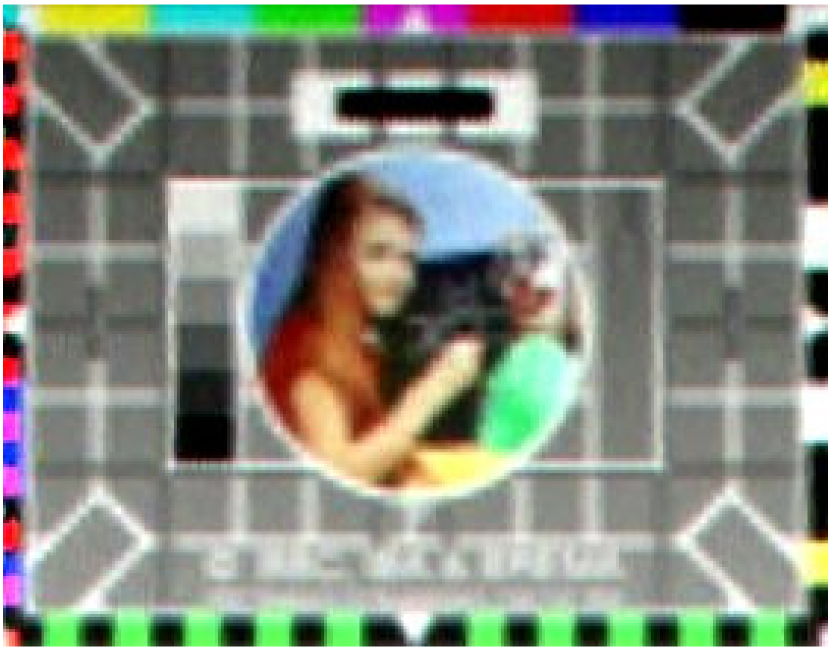


Fig. 37. A 96 x 72 pixel test pattern image

As you can see from Figure 37, image resolution wasn't wonderful, and the frame time in this RGB CO96 mode was nine seconds. However, the system worked remarkably well under difficult NVIS conditions, such as 80 metres at night.

The OFDM NBTV transmit facility was always a separate program. This allowed you to monitor your own transmissions, operate full duplex (send pictures both ways at the same time!), and it allowed 'drag and drop' operation. This way you to send any 4:3 format image from your computer by simply dropping the image on the application. It was also possible to send real-time images from a web cam.

The ZL2AFP OFDM NBTV software has not been widely used, except by ZL2AFP – ZL1BPU, although there has been encouraging feedback from others. There has been at least one other programmer experimenting with this unusual format.

Digital NBTV

The idea of this mode was to avoid the tuning difficulty of the OFDM mode, and its tendency to noisy pictures. Instead it replaced these with a further set of problems!

Apart from the testing phase, I wasn't closely involved in the development, which is mathematically complex, although I suggested using a simplified STANAG 4285 protocol, as I had been studying it at the time.

Digital NBTV[33] mode uses high speed PSK; a system based quite closely on STANAG 4285, and similarly also uses an equaliser system to compensate for path timing variation. Since the image data is digitised, it was possible to add wavelet image correction, and forward error correction. A 1500 Hz 4-PSK carrier is used to send packets containing the image data, the FEC information and a 2-PSK fixed 80-symbol PN sequence header.

The data rate of the Digital NBTV is 1200 baud, and the signal is 2.4 kHz wide. There is a faster, wider option for VHF use, 2400 baud.

[33] http://www.qsl.net/zl1bpu/NBTV/Digital.htm

The source images are always 4:3 aspect ratio, but reduced to 1:1 using interpolation, and further compressed using wavelet compression. The system used Reed Solomon FEC, ideal for a packet-based design.

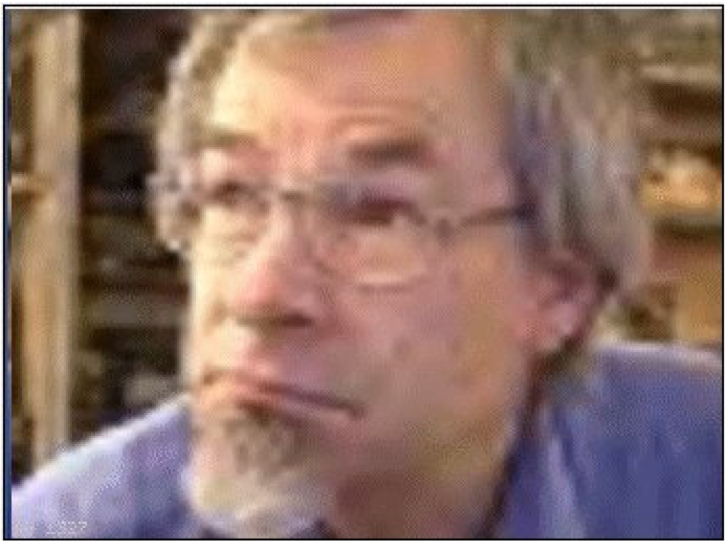


Fig. 38. Digital NBTV reception on 80 metres (yes, that's ZL2AFP!)

As you can see from Figure 38, image quality is good, and noise free. The system has much higher resolution, but is much slower than OFDM Analogue NBTV, although this was kept to a minimum using wavelet compression. A 256 x 341 picture can be sent in 25 seconds.

Con also developed a Hybrid NBTV program[34] (around 2009), which sent analogue data (since it's faster than digital methods), but used a digital PN sequence to achieve frame sync. It worked quite well on 80 metres, but had the usual issues with fades.

Clicklock

Around the same time as the NBTV development (2007), Con developed an adaptation of the G3PLX Clicklock program. This uses a GPS 1pps pulse to compensate for receiver drift, providing phase and timing stability to allow VLF signals to be received and plotted. Because the demodulator was carrier-phase synchronous, with integration over seconds to minutes, sensitivity was extremely high.

Unlike the G3PLX software, which only indicated phase on an integrating Polar plot, the ZL2AFP version also had Cartesian phase and power plots.

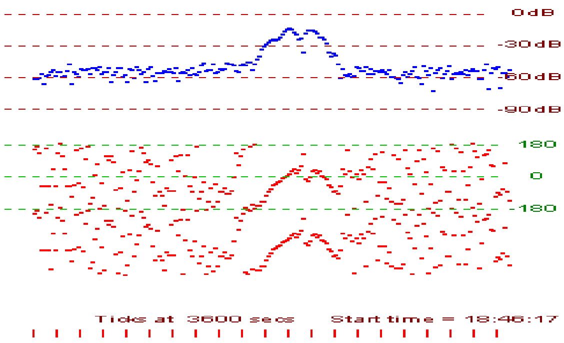


Fig. 39. A Clicklock plot of German station DCF77 (137 kHz) received in NZ

[34] http://www.qsl.net/zl1bpu/NBTV/Hybrid.htm

We used this software to track signals from the USA, Europe and Japan on LF, and Con also received a 10 mW EIRP signal from me on 181 kHz. We were working cross-band at the time, and I was able to change the phase of the transmission (also GPS locked) and have him correctly tell me which direction and how much I had changed the phase.

Nixie Clock

This was intended as a bit of fun, but it certainly looks impressive!

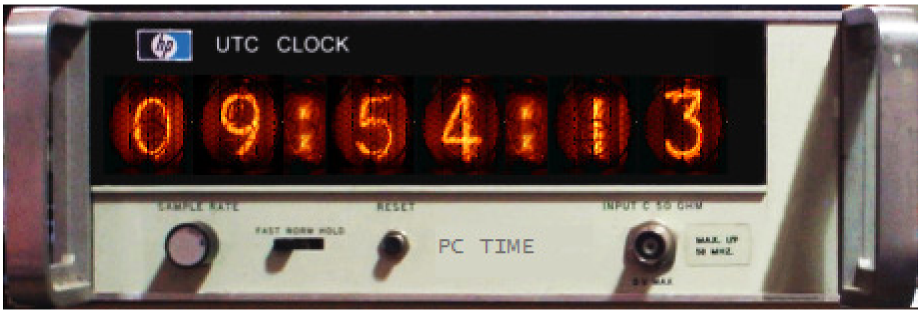


Fig. 40. The Nixie Clock

We designed it (in 2009) to look like a retro Hewlett Packard frequency counter of the 1970s, and it certainly looks realistic. I took photographs of a real counter, which I cut down in width using a photo editor, and I took separate photographs of each of the 10 digit numbers and created a fictitious ‘colon’ image.

Con used the main picture as the program window, superimposed the controls on it, and of course overlaid the time images as necessary. What is really cool is that, as with real Nixie tubes, the digits vary in brightness and move back and forward as the time changes. Extremely realistic!

The Nixie Clock can use computer system time, or can operate from a GPS module using the GGA or RMC serial message, and could show UTC or GPS time. A later version operated from the HP Z3815A GPS Disciplined Oscillator status message.

MSK Experiments

Both Con and I were capable of operating on 134 kHz back in 2007, briefly on 501 kHz around 2009, and nowadays on 474 kHz. Because transmitters for these bands are rarely linear, we were looking for methods of transmission, which were both very sensitive and had no linearity issues. MSK seemed ideal, but receiving it was quite a challenge.

Our first MSK experiments were made around this time, but the details are now rather sketchy. I recall receiving MSK transmissions from MF DGPS stations using MSKRX (G3PLX, 2006). Our plans weren’t to come to fruition until CMSK in 2010 (see later).

PSKSounder

You will recall that we were exploring ideas based on STANAG 4285 (Digital NBTV). Peter G3PLX had in 2003 written a program (SBSTANAG) able to display the time-of-flight of STANAG signals by decoding the PN-sequence header.

We decided in early 2008 to explore the possibility of using frames of our own transmissions, scaled down to suit amateur capability, to explore this same effect. The resulting program, PSKSounder, has two modes. It can transmit standard STANAG 4285 at 9600 bps (1800 Hz carrier), or a slower amateur version at 8000 bps (1500 Hz carrier), which fits within a 2.4 kHz transceiver filter.

Used in the 9600 bps mode, it is able to monitor military STANAG transmissions and measure their path delays (but not of course their traffic).

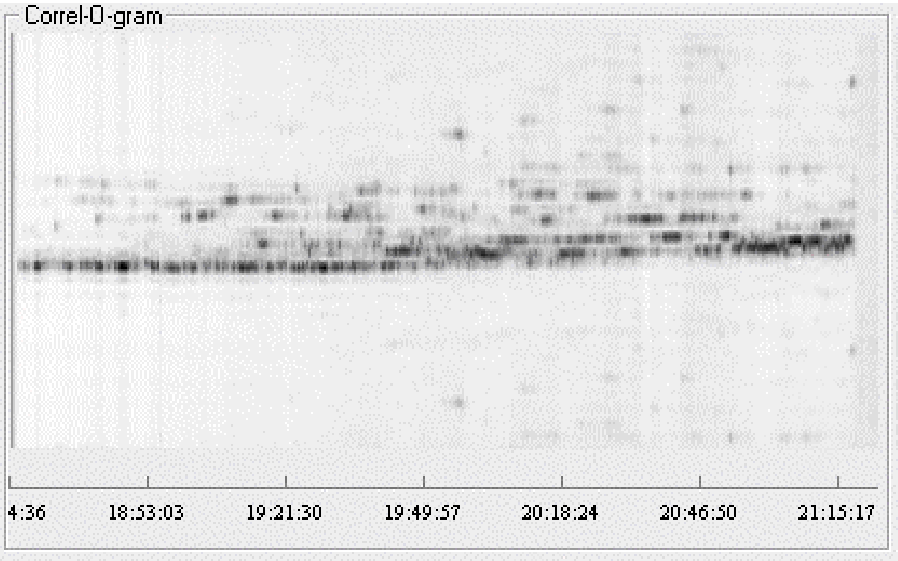
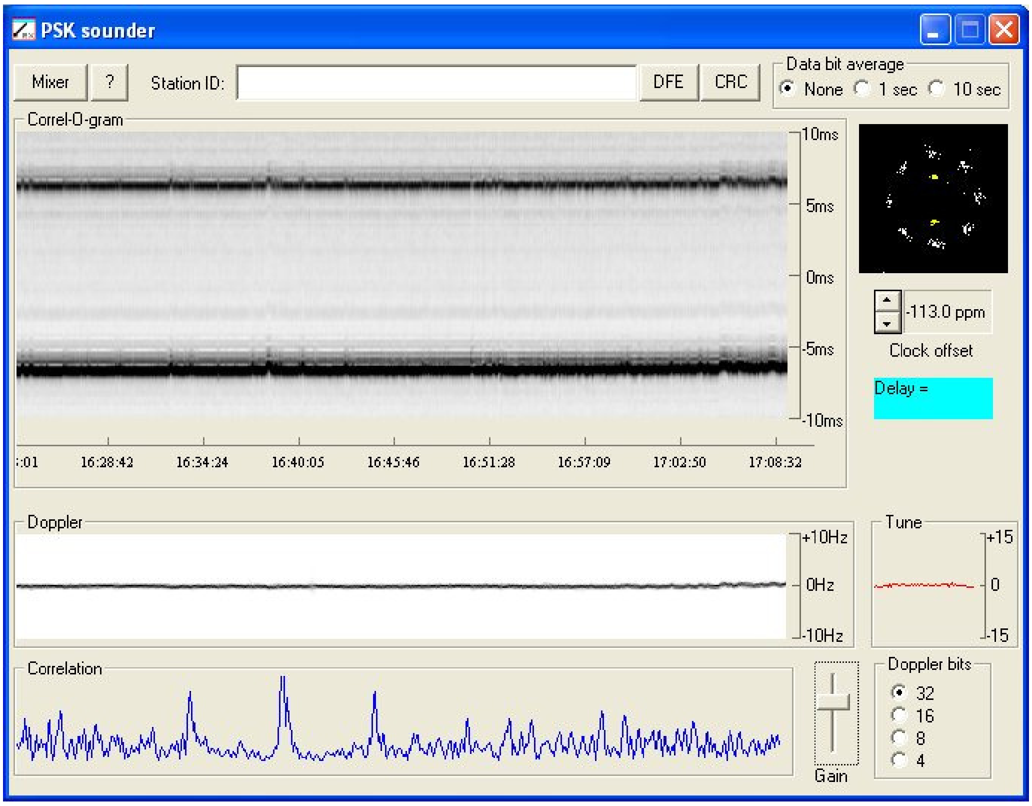


Fig. 41. The ZL2AFP PSKSounder Correlogram

The main feature of the receiving software is a graph of cross-correlator peaks (range 9 ms, the frame period), against elapsed time, which we called a 'Corellogram'. See Figure 41. Timing resolution was about 0.5ms

Con developed a separate transmit program, which generated the standard STANAG header, but didn't replicate the complex STANAG payload. Instead it sends a single repeated ASCII sentence of about 20 characters, with a checksum. The receiver is able to bit-wise average these sentences and display the result when the checksum matches.



Station ID:

Data bit average ● None ○ 1 sec ○ 10 sec

-113.0 ppm Clock offset

Delay =

Doppler bits ● 32 ○ 16 ○ 8 ○ 4

Fig. 42. PSKSounder in action

ZL2AFP Gridmap for WSPR

Also during 2008, Con developed a program that would display received station locators from your WSPR log, on a simple map. Events since, such as the mapping offered by the online WSPR database, have meant that this little program is now much neglected.

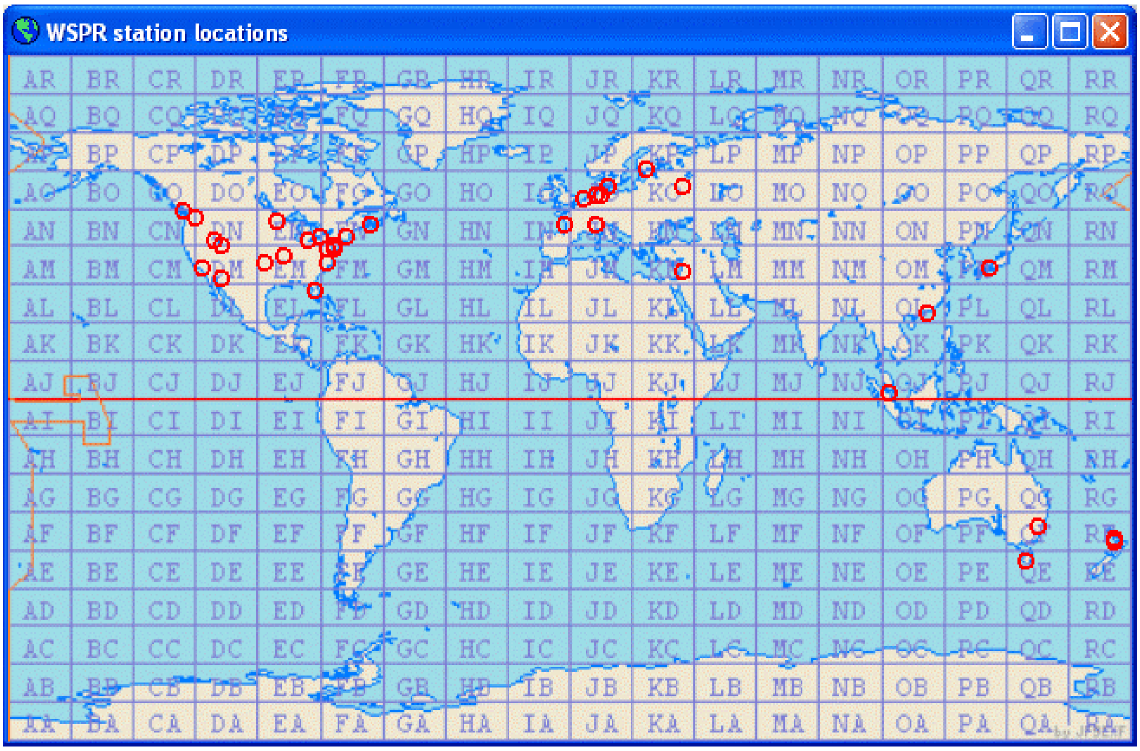


Fig. 43. The WSPR Gridmap

Apart from possibly suggesting the idea, and certainly testing it, I didn't have a lot of input on this occasion. I've since written my own program that uses the online WSPR database to plot propagation for a designated source and destination station over 24 hours. More about that later.

VMEPT[35] Tools

2009 was also when we worked on a series of simple programs to allow the LF/MF and HF amateur to transmit QRP signals that could be visually decoded. The main idea was to implement slow sequential tone multi-tone Hellschreiber, although many more options were possible.

Back in 2003, Marcus Vester DF6NM wrote an interesting DOS program (Chirppix) that would convert a black-and-white picture into MFSK dots. It was very flexible, able to produce sequential and multiplexed output. It could thus be used to transmit crude photographs if the dot rate exceeded the pixel rate.

The next example (Figure 44) shows exactly this effect. The scan rate is set high enough that the image was essentially scanned. I made the image from a photograph, combined with text, saved as a BMP. It was transmitted on 30 metres in October 2014, and received by Peter ZL2IK using Spectrum Laboratory by Wolf Büscher DL4YHF.

The colours aren't part of the original image – they are rendered to indicate signal strength – my shiny forehead shows red!

[35] Visual MEPT, i.e. Manned Experimental Propagation Transmission signals intended to be read by eye

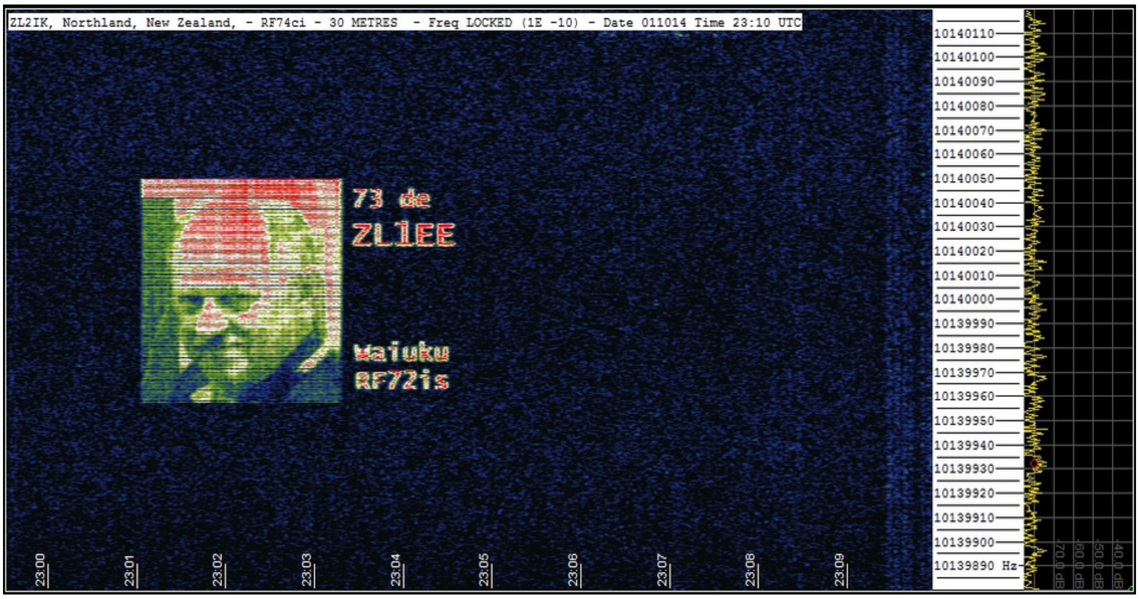


Fig. 44. Chirppix transmission received on a spectrogram.

The versatility of Chirppix allowed it to generate concurrent tones, multiplexed (scanned) images, and also sequential tones. The area that interested me was its ability to generate single-tone pictures. This all depended on the settings. It took a .bmp picture as input, and generated a WAV audio file.

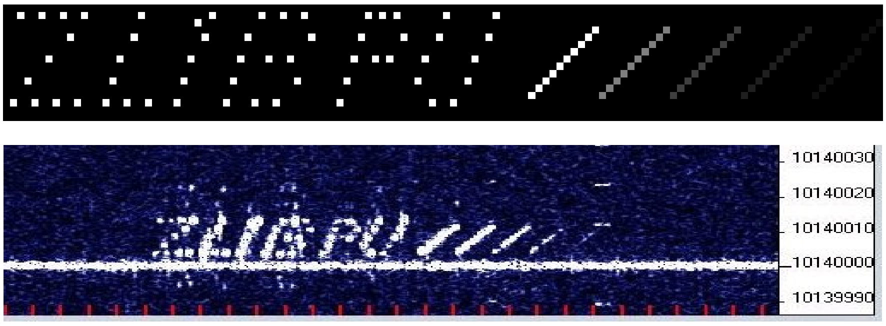


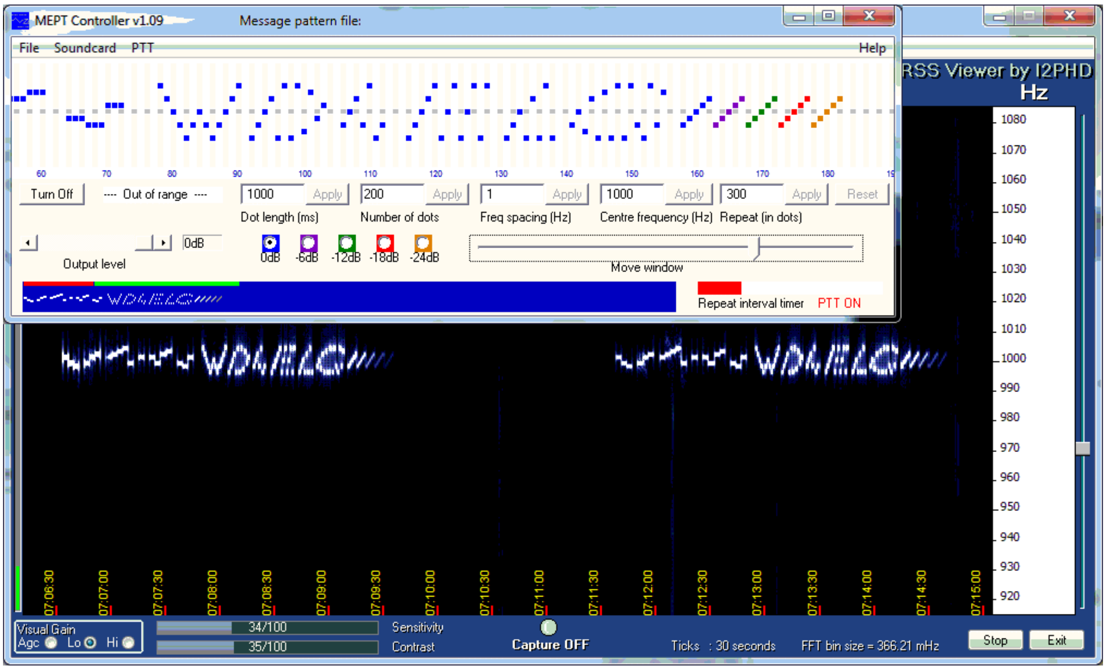
Fig. 45. Source image (above) and resulting signal (below)

The example in Fig 45 (received by VK6DI on 30 metres) shows how accurately the DF6NM Chirppix-generated transmission file matched the input image. The image includes five 6 dB stripes at the right, and you can see all of them on the spectrogram at VK6DI (using Spectrum Laboratory by Wolf DL4YHF).

This was a great system, but making the bit map files was seriously tedious. The simple ZL2AFP VMEPT program simplified transmitting the 'image'. It acted as a WAV player, but with additional functions. It could key the PTT of the transmitter, and repeat the WAV file to a schedule.

The next ZL2AFP Tool for QRSS made generating the patterns so much easier, although you couldn't send photographs. Written in 2012 - 2014, MEPT Controller[36] enabled you to 'draw' the pattern you wanted on a simple graph 16 pixels high, and it would generate a suitable real-time transmission, controlling both the sound card and transmitter PTT. The 2014 program, Version 1.09, can directly transmit audio according to the pattern, or save a .WAV file for later use (Figure 46).

[36] http://www.qsl.net/zl1bpu/SOFT/WYSMEPT.htm



Dot length (ms) 1000

Number of dots 200

Freq spacing (Hz) 1

Centre frequency (Hz) 1000

Repeat (in dots) 300

0dB

Sensitivity 34/100

Contrast 35/100

Ticks : 30 seconds

FFT bin size = 366.21 mHz

Fig. 46. MEPT Controller V1.09, and the resulting output on ARGO

This QRSS[37] MT-Hell concept and MEPT Controller was mentioned earlier under MOSAIC, since this was what started the whole idea, some 15 years earlier. MEPT Controller will transmit anything you can conceive as an image, although it is rather constrained by its sequential (one dot at a time) nature.

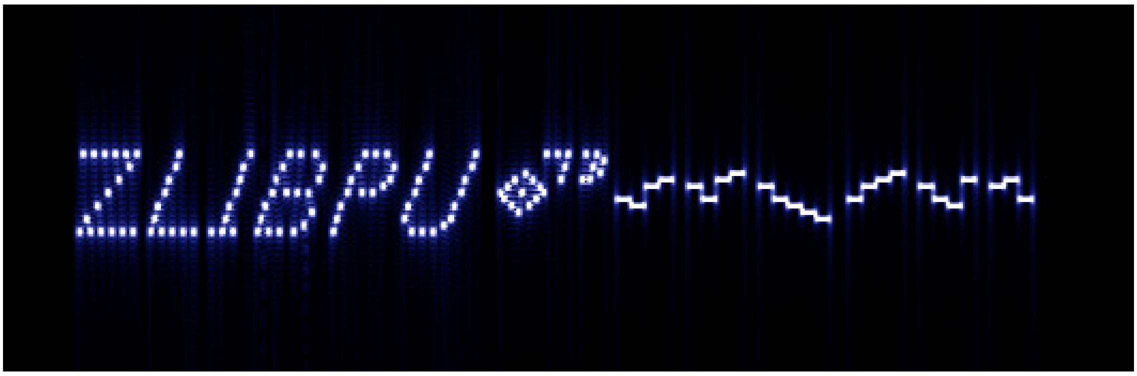


Fig. 47. MEPT Controller will transmit quite a range of images

Nowadays the use of slow sequential MT-Hell is widespread in QRSS applications. It's not uncommon to see it used on HF QRSS channels as well as on 630 metres. The reason for this is the widespread use of 'grabbers' (spectrogram programs which post their captures on a web site every few minutes).

One of the cool features of these grabbers is that many capture at exact five or 10 minute intervals, and if the transmitting station repeats the message at the same rate, a technique called 'stacking' can be used to superimpose multiple images, and in this way enhance the signal and cancel out the noise.

[37] QRS is Q-code for 'send more slowly', so QRSS means 'send even MORE' slowly! Typically this means one symbol every second or slower.

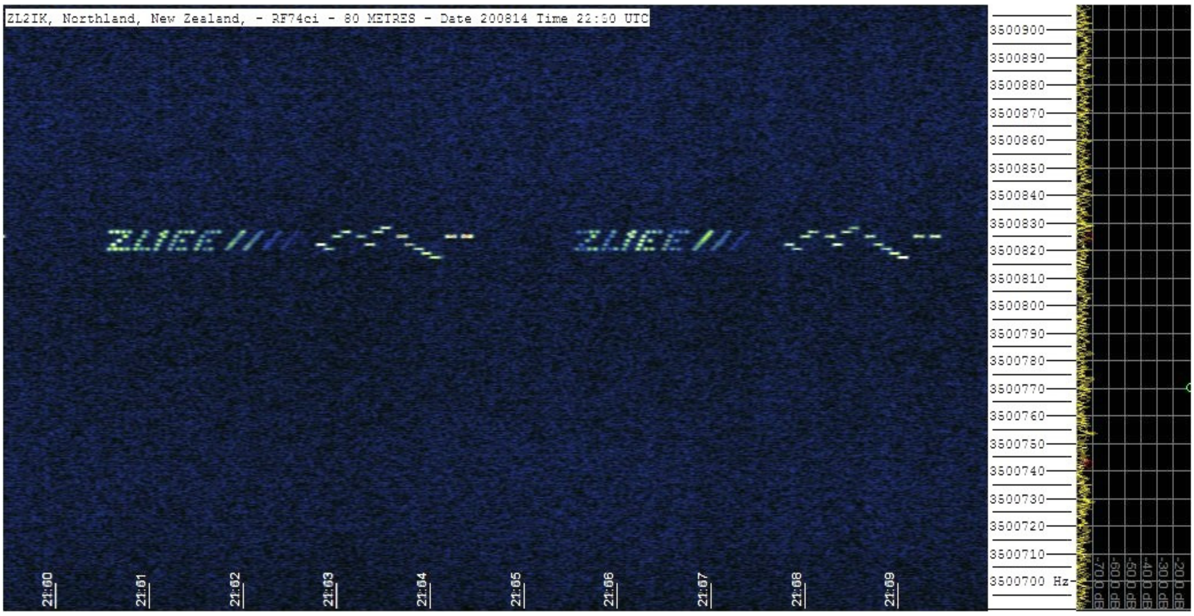


Fig. 48. A ‘stacked’ grabber picture of my 80 metre signal.

Notice in Figure 48 how little noise there is. This was recorded by ZL2IK during the evening, under NVIS[38] conditions, with no doubt plenty of lightning activity (QRN). The frame repeat time is 5 minutes, and the grabber refresh time is 10 minutes. This is a stack of 5 images.

The ‘grabber’ concept was first developed for enhancing astronomy photographs, and also allows objects moving across the static background to be easily detected.

[38] Near Vertical Incidence Signals (typical of ionospheric signals returned at night 2 – 10 MHz)

13. My Own Developments

As you will have guessed, I'm not a real Windows programmer, and don't have any skills in sound card programming. As you have read though, I've been writing programs, mostly in BASIC, for a long time. I mentioned MOSAIC in 1998, but there have been plenty of other projects since. Ages ago I studied computing at University, and also did a series of microprocessor courses. But that was back in the 1970s and 1980s.

AVR Projects

I was an early experimenter with micro-controllers, and developed quite a few AVR based projects in the early 2000s. They were documented on the web site and offered for sale (design free, firmware cost a few $$$). A Digital Voltmeter that was powered from an RS232 port was among the first (2001). In 2003 I developed a Rotator Controller, a GPS referenced clock, and the Minibeacon I mentioned earlier (which would send Hell).

The popular LF Exciter (well, several dozen were built) was first released in 2002, and work on it continued until 2006. It used a software Direct Digital Synthesis (DDS) algorithm pioneered by Jesper Hansen, and it worked quite well up to 500 kHz. The Exciter was quite sophisticated, had 24-bit synthesis for very tiny frequency steps, could be GPS locked, and could be either remote controlled or run off an internal script. It was able to act as a MEPT, and generate quite a few modes autonomously.

Around 2006, and for my own pleasure, I developed a GPS locked frequency standard, with the eventual intention of providing a New Zealand wide broadcast standard on 5 MHz. I developed the GPS decoding section and wrote the firmware to generate the time code. This unit used two ATTiny2313 processors and two LCD displays! The second processor locked a 5 MHz reference and generated the time codes. The two processors communicated on the same serial line as the GPS receiver, and the code was extremely complex. Unfortunately the project went nowhere, as unfortunately my team member Ted Minchin ZL1MT, who was working on the licensing, transmitter and antenna side of things, died soon after.

The most successful of all the AVR programs (and probably the most complex) was the GPS Disciplined Reference, 'Simple GPSDO'. It started life as a project at work, where I used the AVR to lock a 10 MHz OCXO to a much noisier factory-wide 10 MHz reference. I was able to adapt the hardware for my own purposes, using 1PPS from a GPS module as the reference, and completely different code. The processor was a little ATTiny2313.

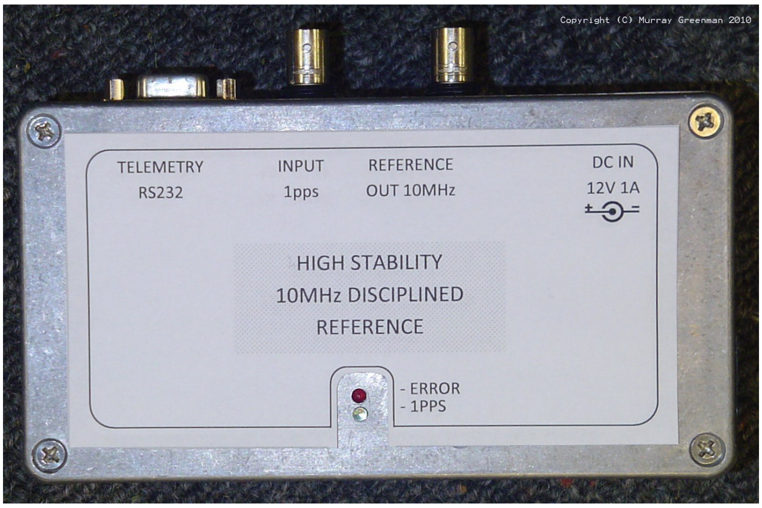


Fig. 49. The Simple GPSDO

The code for a GPS Disciplined Reference is much more complex than the original 10 MHz reference, as you need to provide 'hold over', so the unit will continue to operate correctly when GPS pulses are not available (or not valid), and to recover when the reference returns. The Simple GPSDO[39] development continued from 2007 to 2013, and the code was completely written in assembler. The device achieved a stability of a few parts in 10^{12}. I no longer offer or support AVR projects. I no longer have the tools or the brain for it!

Arduino Projects

I'm just at the beginning of my Arduino career! These little units show great promise for Amateur projects. Both Con and I have experimented with them. My only project to date has been a telemetry collection device that could read voltages, currents, temperatures etc, and pass them in a message frame (via USB) to a PC. The PC (running the FST software) would then format and send these as FSQCall telemetry.[40]

FST (Fast, Simple Telemetry), was one of the few specifically Windows programs I've written. It was written in Just BASIC. I had little choice, since my preferred QB64 compiler does not support serial communications.

CASTLE

One of the problems with QRSS modes is that it can be difficult to fit a whole callsign within a 10-minute grabber frame – especially using on-off keyed Morse. Another problem with on-off keying is that it can be difficult to tell whether a pair of dots is just that, or a dash with a fade in the middle. This is a little better when FSK is used, one frequency for key down, another for key up, but the speed is the same and still a problem.

Various methods have been devised to speed QRSS up. One of them is to use differential keying, dots on one frequency for a dash, and on another for a dot. It was quite difficult to read on a spectrogram. I felt there was a better way. The mode I devised uses multiple tone frequencies, all the same length, with dots one side of an unused centre frequency, dashes the other. From its appearance, like the crenulations on a castle wall, I called the mode CASTLE.[41] The mode has not been widely used, but can easily be generated using most MEPT techniques.

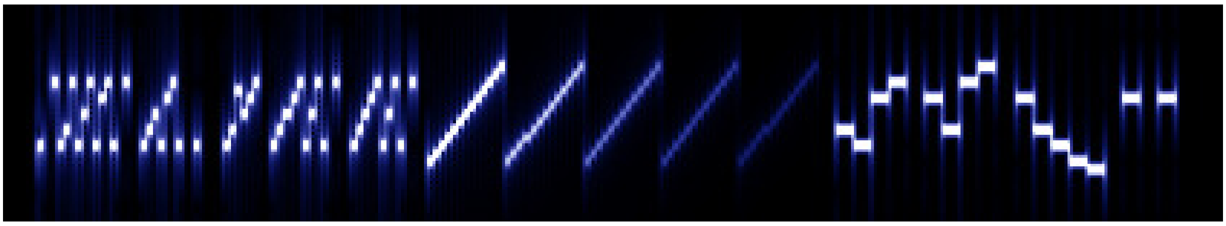


Fig. 50. Multi-mode transmission – 'ZL1EE' in CASTLE on the right.

In the example in Fig. 50, received using ARGO, the dashes are below the centre, the dots above. Within a character, the dots and dashes move out one step (typically 1 Hz) with each repeated same element, and start again near the middle on the next character. Of course the example was sending 'ZL1EE'. The mode is about four times faster than on-off keyed Morse, easier to read in fades, and about 12 Hz wide at most.

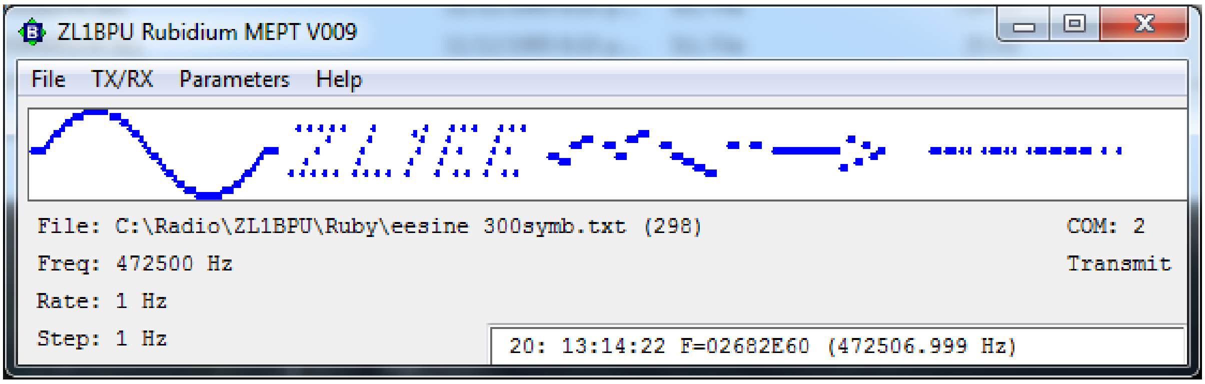
[39] http://www.qsl.net/z/zl1bpu//MICRO/SIMPLE/SimpleGPS.htm
[40] http://www.qsl.net/zl1bpu/SOFT/FSQ%20Telemetry.zip
[41] http://www.qsl.net/zl1bpu/MICRO/EXCITER/castle.htm

QB and QB64

I have been writing programs in the old Quick BASIC for a long time. One of the most notable was a program called Ruby, designed as a controller for the FEI FE-5650 and FE-5680A Rubidium synthesisers. The first version was in 2005, and targeted the HPLX palmtop platform. The HPLX is a tiny DOS PC with limited memory, an 8 MHz processor and a small LCD display. I developed Ruby so that it was able to operate as a MEPT (send patterns in CW, CASTLE or MT-Hell). The most recent version (for PC and HPLX) was written in 2009.

A later program (also 2009), written in Just BASIC, has visual and graphical display of the synthesiser operation (Fig. 51). It is a Windows program with 64-bit compatibility. It has 1 Hz resolution and calculates the synthesiser control values with 32-bit precision. This is especially helpful when used with a high stability synthesiser such as the FEI FE-5650A or 5680A, and for frequency standard transmissions (or frequency measuring contests). This functionality was achieved using just one known command for these synthesisers!

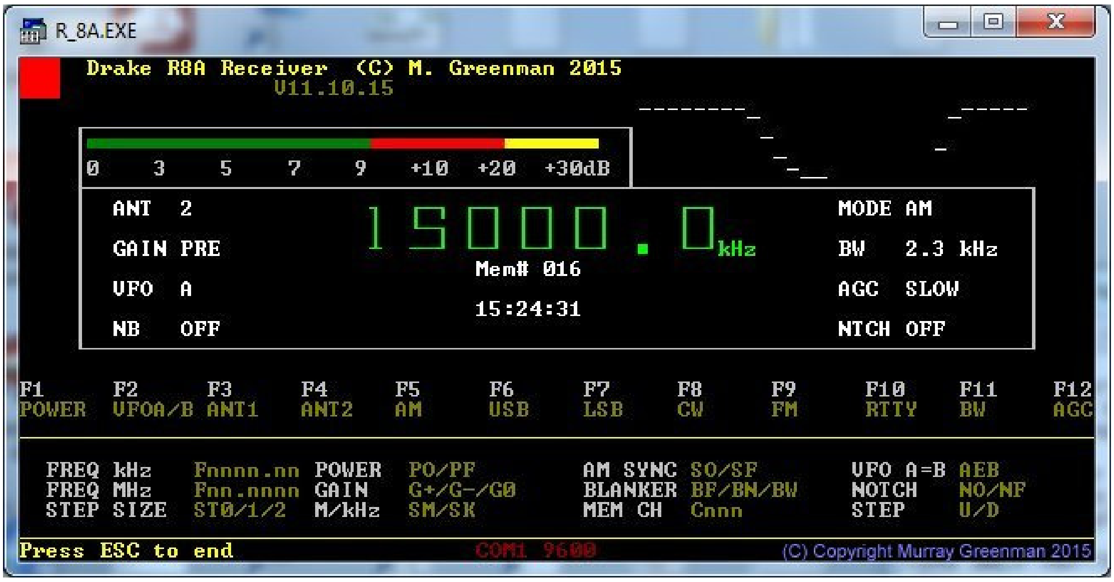
I wrote similar programs for the Novatek 409B and N3ZI DDS synthesisers between 2011 and 2012. The patterns for these controllers are written as simple text files (see later).



```
File: C:\Radio\ZL1BPU\Ruby\eesine 300symb.txt (298)          COM: 2
Freq: 472500 Hz                                               Transmit
Rate: 1 Hz
Step: 1 Hz                        20: 13:14:22 F=02682E60 (472506.999 Hz)
```

Fig. 51. My FE-56xx controller program with MEPT capability.

It is possible to write Quick BASIC and QB64 programs that are Windows compatible, provided you don't use graphics, and don't need 64-bit support. This becomes quite a challenge! In 2015 I wrote a remote controller for the Drake R8A receiver. The program seems to have graphics, but they are all actually written in QB using text block characters.



```
     Drake R8A Receiver  (C) M. Greenman 2015
                      V11.10.15

   0    3    5    7    9   +10  +20  +30dB

     ANT   2                                   MODE AM
     GAIN  PRE        15000.0kHz               BW   2.3 kHz
     VFO   A             Mem# 016              AGC  SLOW
     NB    OFF           15:24:31              NTCH OFF

F1     F2     F3     F4     F5     F6     F7     F8     F9     F10    F11    F12
POWER  VFOA/B ANT1   ANT2   AM     USB    LSB    CW     FM     RTTY   BW     AGC

  FREQ kHz   Fnnnn.nn POWER  PO/PF          AM SYNC SO/SF       VFO A=B AEB
  FREQ MHz   Fnn.nnnn GAIN   G+/G-/G0       BLANKER BF/BN/BW    NOTCH   NO/NF
  STEP SIZE  ST0/1/2  M/kHz  SM/SK          MEM CH  Cnnn        STEP    U/D
Press ESC to end                  COM1 9600        (C) Copyright Murray Greenman 2015
```

Fig. 52. The Drake R8A Controller

In Fig. 52, the real-time S-meter display at the top is a bar graph written using block characters. The large text frequency display is created by strings of line-draw characters. I wrote this program because the one supplied with the receiver was unbelievably poor! It was text-only, and did not poll the receiver status in real time (you could only see a change when you sent a command), and it did not even update the display when you changed a control on the receiver. As you can see, my program displays everything on the receiver front panel, operates in real time, and allows access to all controls.

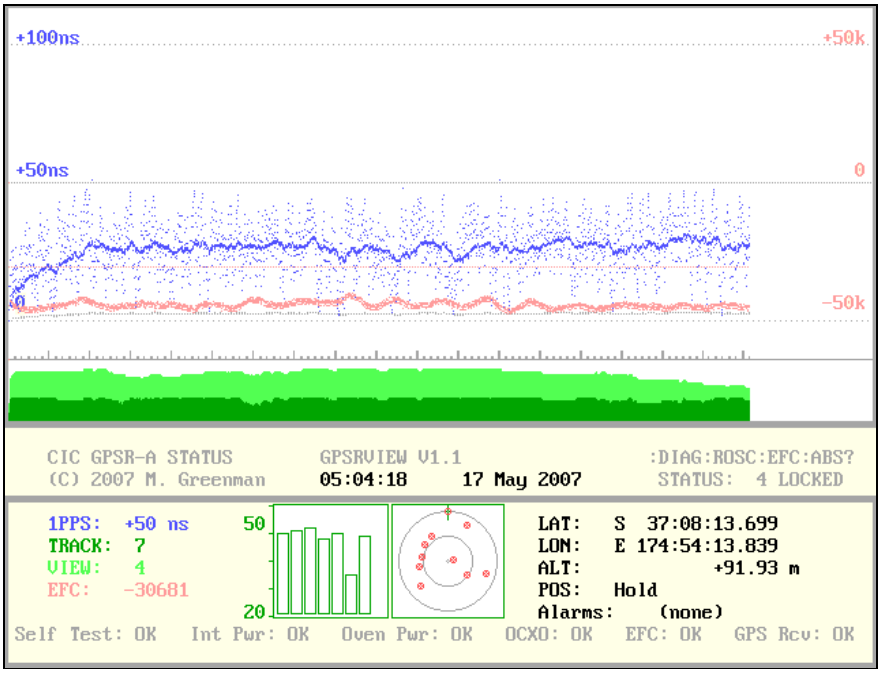
There are limitations to developing programs in BASIC related to modern computers: QB64 will generate 32-bit and 64-bit Windows compatible code, but has no serial communications support: Quick BASIC has excellent serial comms, but no support for Windows (it operates in a DOS box), and will not run at all in 64-bit Windows. The only compiler I have which covers all bases is Just BASIC, which supports 64-bit Windows, but has plenty of other limitations.

GPS and GPSDO programs

Over the years 1999 to 2013 I wrote quite a lot of software for GPS related applications, at first a series of monitors for GPS modules, based on the GPS sentences. Some of these can be quite complex, and it taught me a lot about parsing serial protocols.

I also wrote programs to support commercial GPS Disciplined References, starting in about 2003, after acquiring a Hewlett Packard Z3801A GPSDO. At the time there was very little suitable software available, and what there was, was expensive. Added to the problem was a general lack of information about the communications protocol.

In around 2009 I was able to source quite a few ex-cellular network GPSDOs, the models HP Z3815A, CIC GPRS-A (an HP Z3815 near clone) and the Samsung GCRU-D. This was at the time networks moved to CDMA phones. Most of these units I passed on to friends. I later acquired just one Trimble/Nortel GPSDM. Of course these devices ideally needed monitoring software, the problem being that none of these units had publicly available software. The communications protocols they used were also not available, so I was forced to reverse-engineer the protocol.

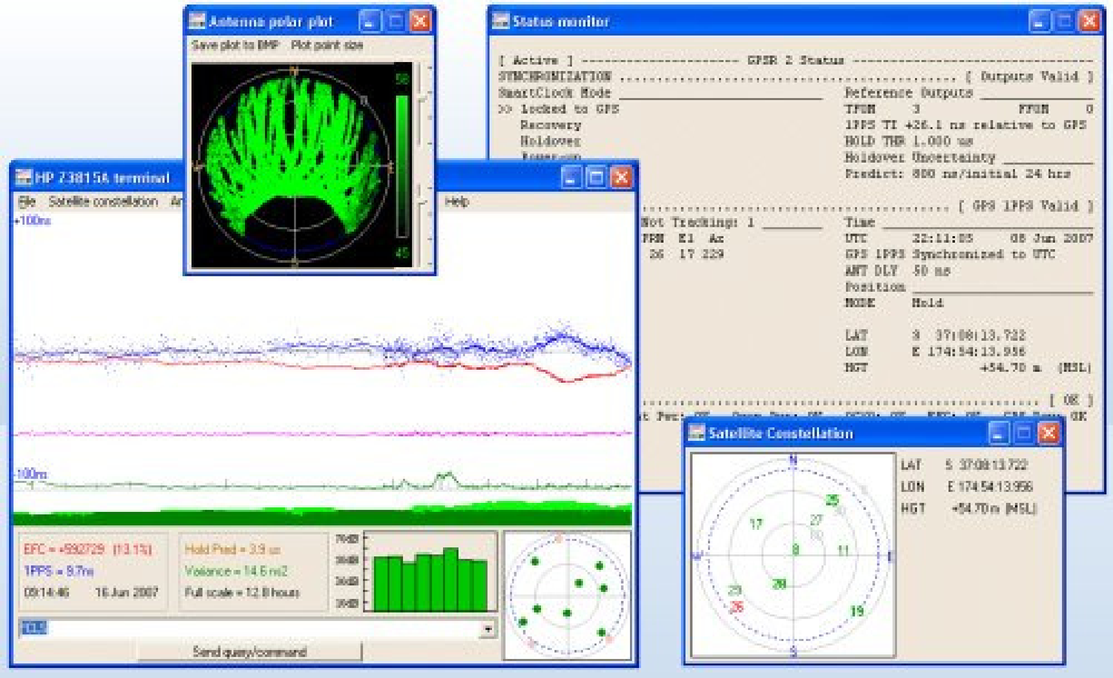


```
CIC GPSR-A STATUS          GPSRVIEW V1.1                :DIAG:ROSC:EFC:ABS?
(C) 2007 M. Greenman       05:04:18     17 May 2007      STATUS:  4 LOCKED

1PPS:   +50 ns                          LAT:    S  37:08:13.699
TRACK:   7                              LON:    E 174:54:13.839
VIEW:    4                              ALT:              +91.93 m
EFC:    -30681                          POS:    Hold
                                        Alarms:     (none)
Self Test: OK   Int Pwr: OK   Oven Pwr: OK   OCXO: OK   EFC: OK   GPS Rcv: OK
```

Fig. 53. My GPSRVIEW software for the CIC GPSR-A

I'd been reverse engineering serial protocols for various equipment, both at work and home since about 1995. Some were very tricky, with different data rates and protocols in the reverse direction, the use of checksums, handshake bytes, rolling codes, you name it! I wrote my own software to discover and then test the GPSDO protocols (as in Figure 53 above). This nice graphical application was written in Quick BASIC. I then passed the protocols and source code on to Con, for him to write Windows versions, which he did for the HP Z3815A and the Samsung GCRU-D. Again, this was in 2009.



```
[ Active ] -------------------- GPSR 2 Status ------------------------------
SYNCHRONIZATION ............................................ [ Outputs Valid ]
SmartClock Mode ________________________  Reference Outputs _____________
>> Locked to GPS                           TFOM      3          FFOM     0
   Recovery                                1PPS TI +26.1 ns relative to GPS
   Holdover                                HOLD THR 1.000 us
                                           Holdover Uncertainty __________
                                           Predict: 800 ns/initial 24 hrs

.................................................... [ GPS 1PPS Valid ]
Not Tracking: 1 _______                    Time _________________________
PRN  El  Az                                UTC       22:11:05    08 Jun 2007
 26  17 229                                GPS 1PPS Synchronized to UTC
                                           ANT DLY   50 ns
                                           Position ______________________
                                           MODE      Hold

                                           LAT       S  37:08:13.722
                                           LON       E 174:54:13.956
                                           HGT                +54.70 m  (MSL)

............................................................... [ OK ]
```

EFC = +592729 (13.1%)
1PPS = 9.7ns
09:14:46 16 Jun 2007

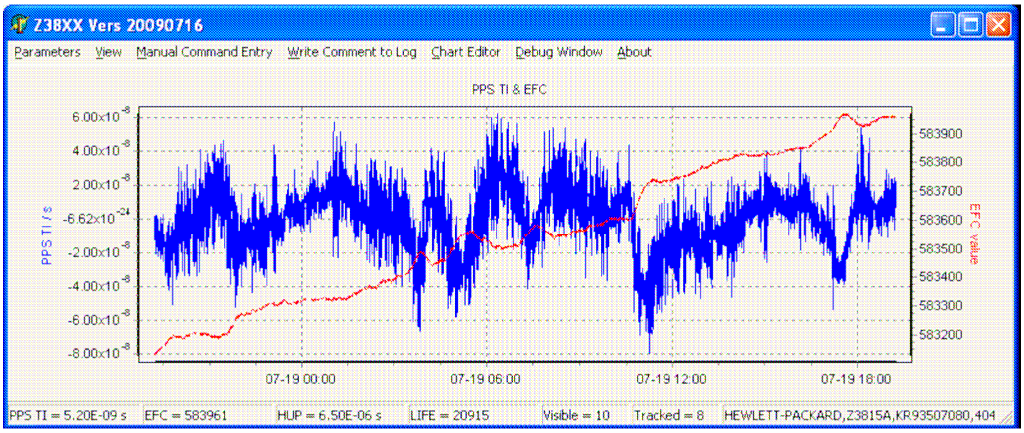
Hold Pred = 3.9 us
Variance = 14.6 ns2
Full scale = 12.8 hours

LAT S 37:08:13.722
LON E 174:54:13.956
HGT +54.70m (MSL)

Fig. 54. The ZL2AFP software for the Samsung GCRU-D

This massive reverse-engineering effort resulted in a really super program, Z38XX, by Ulrich Bangert DF6JB.[42] It was written in 2011, originally for just the Z3801A. I contacted him and a fruitful exchange followed. I sent him the commands I knew for the Z3815A and Samsung GCRU-D. He expanded it using the information I sent him, and it was last updated in 2013.

I wrote the English language manual for his program in 2011, and at the time, the software supported the Z3801A, the Z3805A, the Z3815A, the Jackson Firefly and Ruby, and the Samsung GCRU-D.



PPS TI = 5.20E-09 s | EFC = 583961 | HUP = 6.50E-06 s | LIFE = 20915 | Visible = 10 | Tracked = 8 | HEWLETT-PACKARD,Z3815A,KR93507080,404

Fig. 55. The main window of Z38XX

[42] http://www.ulrich-bangert.de/Z38XX.zip

I never successfully wrote software for the Trimble/Nortel GPSDM, although I gave it a good shot! Nobody else I know of has one of these, and the communications protocol used is peculiarly dense. Fortunately I was able to score some original Nortel software and manuals for this remarkable unit. I simply contacted a technician at Trimble Australia to scrounge the official tools.

The Script Concept

Back in 2005, as already mentioned, I wrote a simple control program for the FEI-5650A Rubidium Synthesiser, which was compiled to operate on the HPLX1000 palm-top computer. Soon after I decided to add a level of automation that would allow the synthesiser to operate as a MEPT. I developed a simple ‘script’ idea, where the necessary commands were written as a text file, and the control program would interpret the commands and send them to the synthesiser in its own command language. The FEI synthesisers have only one command: ‘F=xxxxxxxx’ in hexadecimal! Others have more commands, but still just a few.

The script idea from 2006 grew, and by 2009 I had controllers for several kitset and commercial synthesisers written in JustBASIC (Figure 51 is an example). In 2011 I persuaded Tom Baier DG8SAQ to include script capability (using my definitions) in his Si570 controller. The idea was included in JASON by Alberto I2PHD and in WSQ by Con ZL2APF and later WSQ2 by Wolf DL4YHF, although in these a separate script ‘compiler’ is required.

The ZL1BPU LF Exciter had the ability to follow script commands, and actually had native script capability reading from EEPROM. It had extra commands for mode and power level.

The script files have only three lines. Here’s an example, which sends ‘BPU’ in QRSS3 Sequential MT-Hell:

```
3
0.5
S06789A68A68A68A79XX6789A8XA8XA9XX6789A6X6X6789AXXXX
```

The first line sets the default tone duration (3 seconds for QRSS3). The second sets the tone spacing in Hz, and the third line is the actual script. There are just three script commands:

0 – 9, A – F	Tone to be transmitted, one of 16, with pre-defined equal spacing.
Sn	Speed (duration of tone). This is a multiplier, n+1, default 1.
Q	Exits the pattern generation after one iteration.
X	Transmitter off for one element duration.

The operating frequency is preset prior to starting the script, and is the frequency of the ‘8’ command in the script. The S command is used to simplify multi-mode scripts, so that (for example) Morse dashes could be sent as a single command, by sending ‘S28’.

The Q command enabled script interpreter programs to be called by timer and send the message just once. Without it, the script repeated over and over.

The X command tells the synthesiser to operate at zero frequency, so turns off completely for one element period. This allows the synthesiser to generate CW Morse as well as complex MT-Hell patterns such as that in Figure 51.

Some synthesisers (e.g. the Novatech 409B) also have a Pn command, which sets power level.

14. CMSK

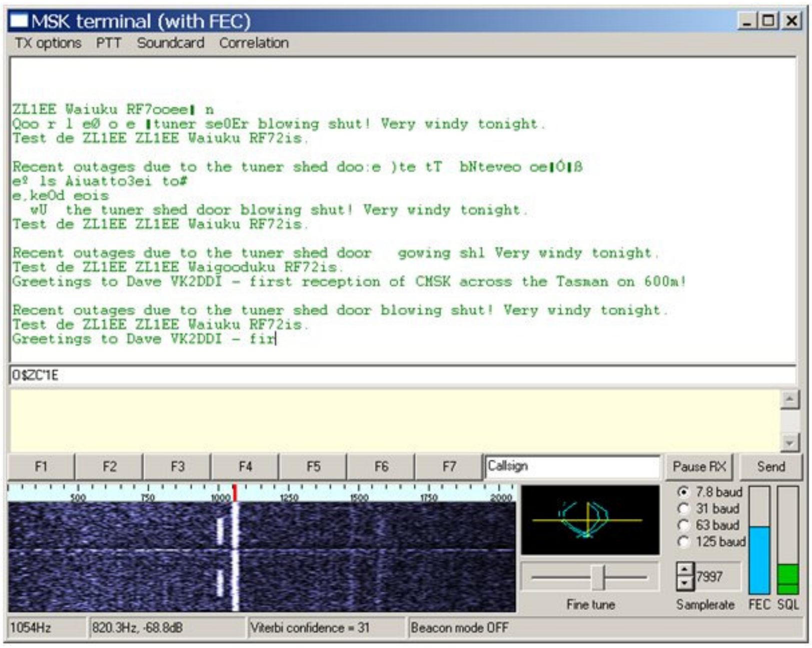
As I mentioned earlier, Con and I have had an ongoing interest in using MSK, as it offers specific benefits on LF/MF. It doesn't need a linear transmitter, is narrow band, and therefore sensitive. But it can be tricky to tune, and sync recovery can be a challenge.

We had also been keen to mess with P-N sequences (as we had in PSKSounder), using the sequence to time (and sync) locate a data frame and also to provide fine-tuning. We called the scheme we came up with CMSK. It probably should have been C^{3}MSK, since the name meant 'Correlated, Convolved, Chat-mode MSK' – quite a mouthful.[43] This was developed in 2011. It was the first amateur chat mode to use MSK, and the first to use a P-N sequence convolved with the data (WSPR does this, but isn't a chat mode, and uses fixed frame timing).

We used a P-N sequence to anchor the frame, but in addition it also provided the symbol sync. It used convolutional FEC with an interleaver (NASA R=1/2, K=7) and a Viterbi decoder. The P-N sequence was recovered using a cross-correlator. The software was able to lock to the P-N sequence even when it was too weak to reliably decode.

Interleavers are often a problem to synchronise. We chose to use a matrix interleaver, which matched the frame size, and to transmit the P-N sequence spread along the length of the frame (i.e. *convolved* with the data). It's a bit too complex to describe here, but it is a cunning system.

CMSK uses an alphabet similar to PSK31, which was extended to provide secondary text. The default 63-baud mode operates at 30 WPM, and is 100 Hz wide.



```
ZL1EE Waiuku RF7ooee▌ n
Qoo r l eØ o e ▌tuner se0Er blowing shut! Very windy tonight.
Test de ZL1EE ZL1EE Waiuku RF72is.

Recent outages due to the tuner shed doo:e )te tT  bNteveo oe▌Ó▌ß
eº ls Aiuatto3ei to#
e.ke0d eois
  wU  the tuner shed door blowing shut! Very windy tonight.
Test de ZL1EE ZL1EE Waiuku RF72is.

Recent outages due to the tuner shed door   gowing shl Very windy tonight.
Test de ZL1EE ZL1EE Waigooduku RF72is.
Greetings to Dave VK2DDI - first reception of CMSK across the Tasman on 600m!

Recent outages due to the tuner shed door blowing shut! Very windy tonight.
Test de ZL1EE ZL1EE Waiuku RF72is.
Greetings to Dave VK2DDI - fir
```

O$ZC'1E

● 7.8 baud ○ 31 baud ○ 63 baud ○ 125 baud

Samplerate 7997

1054Hz | 820.3Hz, -68.8dB | Viterbi confidence = 31 | Beacon mode OFF

Fig. 56. VK2DDI receiving ZL1EE CMSK on 472 kHz

Despite the efforts we went to, to provide good metrics (note the eye diagram in the picture above), the mode proved to be quite difficult to use, and it was soon overtaken by other events, specifically WSQ, which was even more sensitive, and a lot easier to use.

[43] http://www.qsl.net/zl1bpu/CMSK/cmsk.htm

15. WSQ Development

What started this major effort was an interest in (but dissatisfaction with) JASON, by Alberto I2PHD. There were several good ideas in JASON, which we held onto, but abandoned others. JASON worked well, was excellent for LF and MF, but was extremely slow. It operated at only 0.4 WPM.

Some of this slowness was due to the low symbol rate, 0.08 baud, but it didn't help that every character required two symbols.

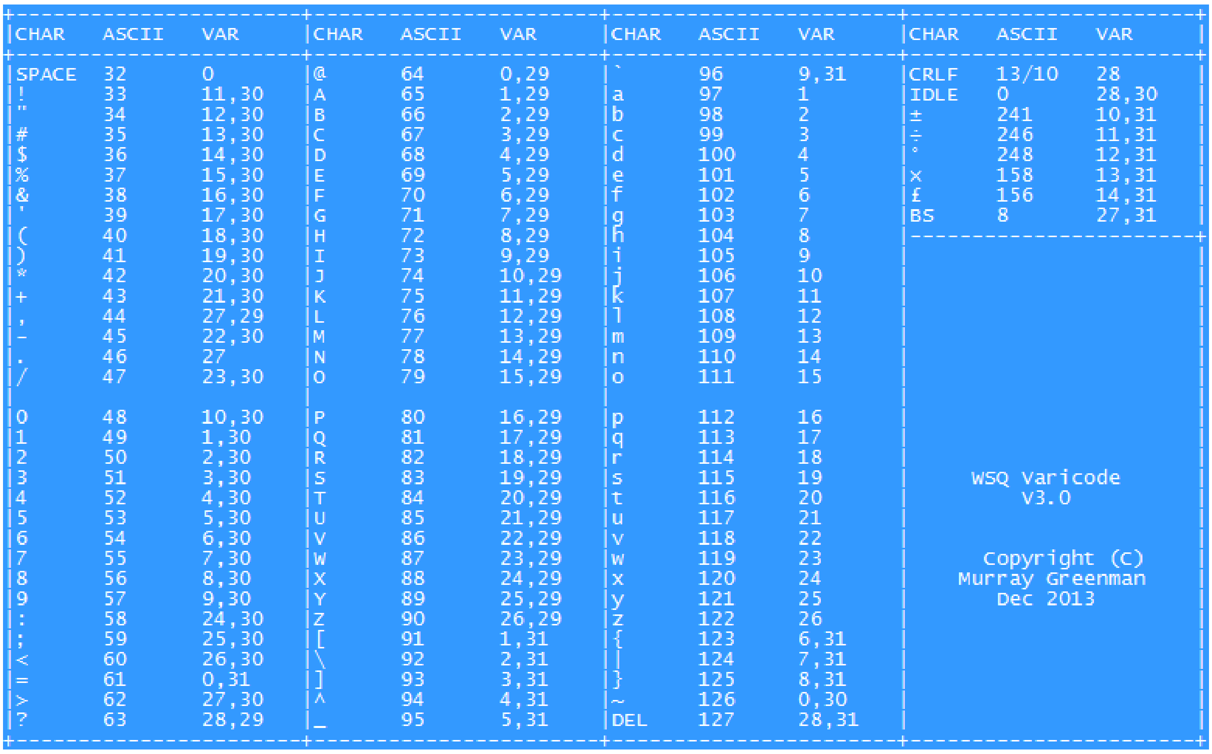
Alphabet Coding

The limitation was to do with the alphabet coding. JASON sent 7-bit ASCII characters in two sequential symbols, four bits (a nibble) each. There was no varicoding. I thought we could do better, so I devised the now famous WSQ Varicode Alphabet.

In those days the conventional thinking (such as it was) held that multi-symbol non-binary alphabets (at least before DominoEX), should be 'square', i.e. have x columns of x characters, which clearly gives a maximum number of characters, x^2.

The DominoEX varicode was a first departure from this thinking, as it used up to three symbols per character, and had 4 x 8 x 8 arrangement, with a maximum of 256 characters, including a secondary character set.

But the WSQ varicode was to be a lot simpler:



CHAR	ASCII	VAR	CHAR	ASCII	VAR	CHAR	ASCII	VAR	CHAR	ASCII	VAR
SPACE	32	0	@	64	0,29	`	96	9,31	CRLF	13/10	28
!	33	11,30	A	65	1,29	a	97	1	IDLE	0	28,30
"	34	12,30	B	66	2,29	b	98	2	±	241	10,31
#	35	13,30	C	67	3,29	c	99	3	÷	246	11,31
$	36	14,30	D	68	4,29	d	100	4	°	248	12,31
%	37	15,30	E	69	5,29	e	101	5	x	158	13,31
&	38	16,30	F	70	6,29	f	102	6	£	156	14,31
'	39	17,30	G	71	7,29	g	103	7	BS	8	27,31
(	40	18,30	H	72	8,29	h	104	8			
)	41	19,30	I	73	9,29	i	105	9			
*	42	20,30	J	74	10,29	j	106	10			
+	43	21,30	K	75	11,29	k	107	11			
,	44	27,29	L	76	12,29	l	108	12			
-	45	22,30	M	77	13,29	m	109	13			
.	46	27	N	78	14,29	n	110	14			
/	47	23,30	O	79	15,29	o	111	15			
0	48	10,30	P	80	16,29	p	112	16			
1	49	1,30	Q	81	17,29	q	113	17			
2	50	2,30	R	82	18,29	r	114	18			
3	51	3,30	S	83	19,29	s	115	19			
4	52	4,30	T	84	20,29	t	116	20			
5	53	5,30	U	85	21,29	u	117	21			
6	54	6,30	V	86	22,29	v	118	22			
7	55	7,30	W	87	23,29	w	119	23			
8	56	8,30	X	88	24,29	x	120	24			
9	57	9,30	Y	89	25,29	y	121	25			
:	58	24,30	Z	90	26,29	z	122	26			
;	59	25,30	[	91	1,31	{	123	6,31			
<	60	26,30	\	92	2,31	|	124	7,31			
=	61	0,31	]	93	3,31	}	125	8,31			
>	62	27,30	^	94	4,31	~	126	0,30			
?	63	28,29	_	95	5,31	DEL	127	28,31			

WSQ Varicode
V3.0

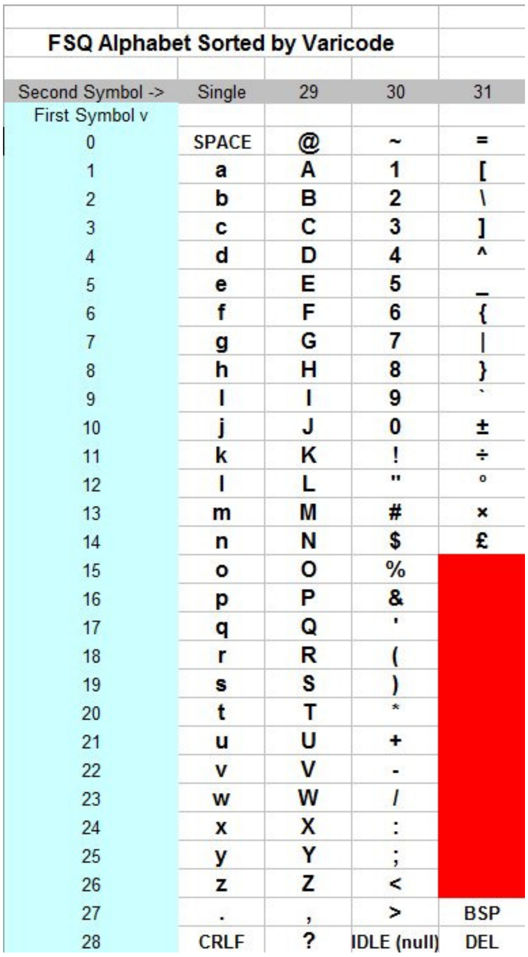
Copyright (C)
Murray Greenman
Dec 2013

Fig. 57. The WSQ Varicode alphabet, used by WSQ, FSQCall and later WSQCall.

It occurred to me then that this thinking was not now relevant, especially in a system with a large number of tones, where you could achieve a huge single- and two-symbol alphabet quite readily. So, for the first time ever, I devised the WSQ alphabet (Fig. 57) to be 29 x 4, i.e. highly asymmetrical. This is more easily seen in Fig. 58. There are 116 possible character codes, although not all are currently used (the red area).

WSQ is based on a system with 33 tones. The alphabet specifies all the lower case characters, plus often used characters (CRLF, SPACE, PERIOD) to be sent as a single symbol (the 'Single' column in Fig. 58). It virtually quadrupled the speed compared with JASON: half as many symbols, and most characters just one symbol. But it needed its own program, capable of sending and receiving 33 tones.

We started with WSQ1, written in ANSI C, but never released. This worked at 0.5 baud, and used the new WSQ varicode alphabet. It had some advanced features at the time, including a noise blanker and noise reduction. It operated as a conventional chat mode.



FSQ Alphabet Sorted by Varicode

Second Symbol ->	Single	29	30	31
First Symbol v				
0	SPACE	@	~	=
1	a	A	1	[
2	b	B	2	\
3	c	C	3	]
4	d	D	4	^
5	e	E	5	_
6	f	F	6	{
7	g	G	7	|
8	h	H	8	}
9	I	I	9	`
10	j	J	0	±
11	k	K	!	÷
12	l	L	"	°
13	m	M	#	×
14	n	N	$	£
15	o	O	%	🟥
16	p	P	&	🟥
17	q	Q	'	🟥
18	r	R	(	🟥
19	s	S	)	🟥
20	t	T	*	🟥
21	u	U	+	🟥
22	v	V	-	🟥
23	w	W	/	🟥
24	x	X	:	🟥
25	y	Y	;	🟥
26	z	Z	<	🟥
27	.	,	>	BSP
28	CRLF	?	IDLE (null)	DEL

Fig. 58. The WSQ alphabet sorted by Varicode

The improved version WSQ2 was released in December 2013. It is also a conventional chat mode. WSQ2 uses 0.512 baud, 33 tones, with the tones spaced 4/T (four times the symbol rate, or 2.048 Hz). Modulation was of course IFK+, but this time with an offset of just 1, rather than 2 as in DominoEX. For the first time, WSQ2 used CAT transmitter control, and it also allowed frequency control commands to be sent via a serial port, in order to control a DDS synthesiser. At 0.5 baud it achieved almost 5 WPM (a mere 12 x faster than JASON!).

Symbol Sync

For WSQ2, we borrowed from JASON the 'peak hits' idea for determining symbol sync. It worked very well, and we thought at the time that it was just something that suited LF/MF. We were to discover otherwise over the next couple of years.

The 'peak hits' algorithm examined the results from the FFT demodulator, running typically at 12 or 16 times the symbol rate. When the same bin had the biggest peak for three successive samples, different from before, this was assumed to be a new symbol. When the peak bin subsequently changed, the previous peak was then deemed to be a symbol, and thus decoded.

The process is asynchronous, so is relatively independent of the transmit symbol timing; we found that the algorithm has a reliable symbol rate decoding range of 6:1! This means that the transmitting station could change speed without needing to inform the receiver, and the receiver needs only one setting, which proved very convenient.

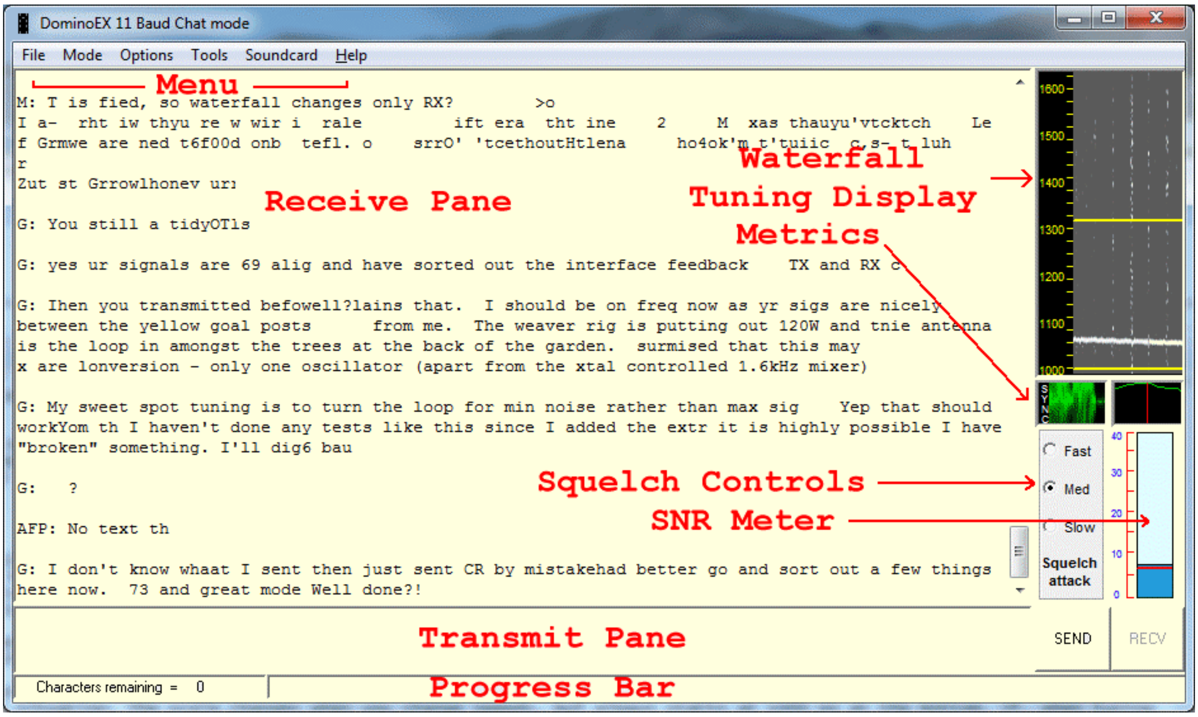
No symbol sync is transmitted in most MFSK modes, and as has been explained, previous sync recovery methods were usually based on a triangular FFT energy envelope recovery and phase locked loops. This method is reasonably noise-immune, but is impractical below about 8 baud, as the loop lock-up time becomes excessive. Prior to this 'peak hits' discovery by I2PHD, operating at 0.5 baud would not have been possible. Now we even use 0.25 baud at times!

Chat vs Sentence Mode

Time for an explanation, as this topic will come up again. Most digital modes allow you to type as you transmit. Some start transmitting as soon as you type; others allow you to 'type ahead', so you can start assembling what you want to say, and the program won't transmit until you press the TX button. You can continue to type while transmitting. We (well, Con and I) call this 'chat mode'.

At about this time (early 2014) I asked Con to revisit DominoEX, to see if it was practical to operate in what I call 'sentence mode'. This had no effect on the receiver, but meant that the text you typed stayed in the buffer untransmitted until you pressed 'ENTER' at the end of the sentence. This is precisely the same approach as Internet chat and telephone text messaging, and I felt it would be a good model for later development (see later – FSQCall).

Sentence mode also encouraged sending shorter transmissions, which I've always preferred, and this becomes really important at very slow transmission rates.



```
M: T is fied, so waterfall changes only RX?        >o
I a-  rht iw thyu re w wir i  rale         ift era  tht ine    2     M  xas thauyu'vtcktch    Le
f Grmwe are ned t6f00d onb  tefl. o    srrO' 'tcethoutHtlena     ho4ok'm t'tuiic  c s- t luh
r
Zut st Grrowlhonev ur:

G: You still a tidyOTls

G: yes ur signals are 69 alig and have sorted out the interface feedback    TX and RX c

G: Ihen you transmitted befowell?lains that.  I should be on freq now as yr sigs are nicely
between the yellow goal posts      from me.  The weaver rig is putting out 120W and tnie antenna
is the loop in amongst the trees at the back of the garden.  surmised that this may
x are lonversion - only one oscillator (apart from the xtal controlled 1.6kHz mixer)

G: My sweet spot tuning is to turn the loop for min noise rather than max sig    Yep that should
workYom th I haven't done any tests like this since I added the extr it is highly possible I have
"broken" something. I'll dig6 bau

G:   ?

AFP: No text th

G: I don't know whaat I sent then just sent CR by mistakehad better go and sort out a few things
here now.  73 and great mode Well done?!
```

Characters remaining = 0

Fig. 59. The introductory screen for DominoEXChat

Although the test software (called DominoEXChat) was never widely released, we used it between us for a while, and although there were grumbles at first, the idea worked well. The program also suggested future experiments with CSMA (Carrier Sense Multiple Access), so that the program should not transmit if the squelch was open. Otherwise, it was just the same as DominoEX, and was interoperable with users of conventional DominoEX.

Although there is a change in mind set required regarding operating procedure, I was very pleased with the results. The idea I had in mind back then was that by employing sentence mode, I could later specify a protocol that could be used to automatically identify stations and provide automated functions - as proved to be the case.

More WSQ2

WSQ2 proved to be quite popular among the European LF/MF set, as it provided really good sensitivity (around –23 dB SNR), wasn't too slow (about 5 WPM), and was easy to use.

Wolf DL4YHF took the idea and made an even better program out of it. When we revisited WSQ2 recently (2017) and improved the modem, Wolf also swung into action and supported the new arrangement. However, Wolf's version remains a conventional chat mode.

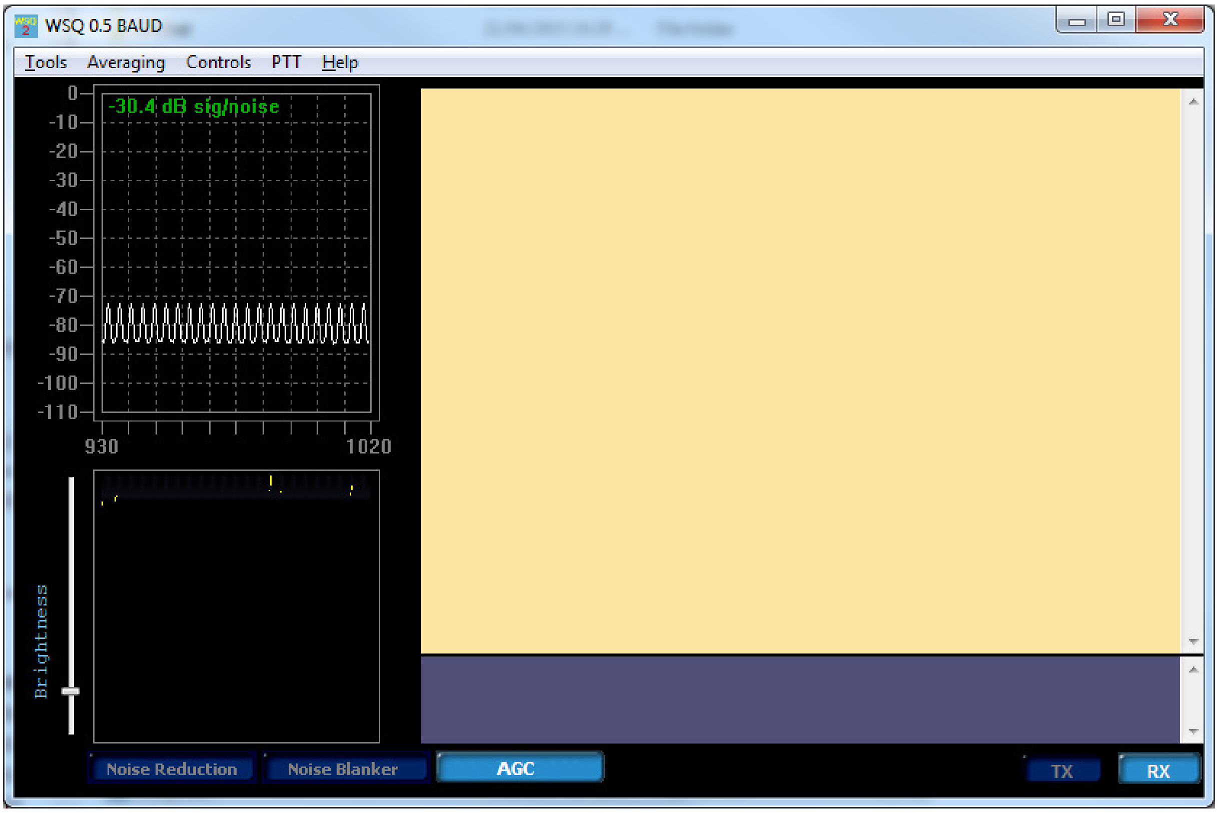


Fig. 60 The simple ZL2AFP WSQ2 user interface

15. FSQ Development

We were so pleased with how WSQ2 worked, that we thought we should try an HF version. This was partly because we wanted to see how the ‘peak hits’ decoder worked with incoming ionospheric timing variation. By now I was also rather dissatisfied with the performance of existing modes for low HF (particularly the NVIS bands, for example 80 and 40 metres). The best at that time was DominoEXChat, but I thought there was room for improvement, through careful control of the spectrum and the use of lower symbol rates.

The first experimental FSQ version was written in late 2014. It had a choice of 3/T and 4/T symbol spacing (so we could experiment) and operated between 3 and 8 baud. It had sentence mode right from the start, but was otherwise a sped-up version of WSQ2.

We very carefully tested the new modem using both an ionospheric simulator and my MEPT. This is an automated transmitter that is able to send over and over, using WAV files of various transmission candidates. By sending the different candidates one after the other on NVIS paths on different bands for several weeks, we were able to get a very good picture of which parameters (baud rate, tone spacing) gave the best results. There were several test programs with different modem candidates that were tested in this way.

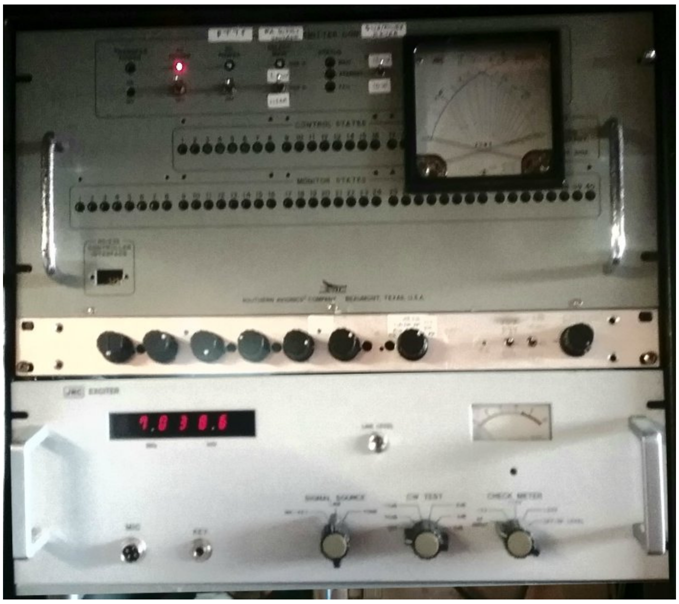


Fig. 61. My HF MEPT

The MEPT transmitter consists of a JRC SSB Exciter with 100 W PA, homebrew controller and power supplies. Messages are transmitted as audio files from a Linksys NSLU2 running embedded Linux. The transmitter operates 1.8 to 30 MHz.

It was very obvious from these tests that the best modem candidates were not only very robust, but the throughput at the chosen symbol rates, especially the later default 4-baud rate, was truly excellent. Achieving 44 WPM at just 4 baud is certainly remarkable.

> *(As an aside, the ‘quick brown fox’ sentence requires around 500 symbols in Morse, 480 in RTTY, yet can be sent in just 69 symbols in FSQ or WSQ. This gives some idea of the improvement in character coding efficiency achieved with this remarkable alphabet.)*

The tests also vindicated the design for NVIS performance – slow symbol rate to provide noise rejection and sensitivity; narrow tone spacing and multiple tones to provide maximum intelligence per symbol and per Hz bandwidth; highly efficient alphabet coding for maximum throughput, IFK+ providing excellent tolerance of drift, the ability to handle narrow tone spacing, mistuning and Doppler shift; and of course most important of all, spectrum management to counter inter-symbol interference.

No other mode would give such robust performance without error correction. We are firmly of the opinion that if you get the modem right in the first place, FEC is needed only to correct genuine errors, not compensate for modem limitations.

Tone Spacing

One aspect that's not been mentioned so far is the business of tone spacing relative to the FFT bin spacing. As we showed in MFSK16, you can operate an FFT demodulator at 1/T spacing, the same as in PICCOLO. 1/T spacing is the closest you can place the tones and still achieve orthogonality. But it leaves you rather prone to Doppler errors. In WSQ at first we used 4/T, which gave much better distance, at a slight expense in signal to noise ratio.

For FSQ we experimented with 4/T and 3/T, and came to the conclusion that 3/T was quite enough. Even better, it provided perfect removal of Doppler and drift errors, since an error of one bin in the IFK+ differencing algorithm would be rounded out. With a bin resolution three times higher than the tone spacing, the differencing algorithm also needed to divide by three. We have stayed with 3/T since, including in WSQCall.

Selective Calling

Plain FSQ was never released. We used it only as a development tool. It had a user interface very similar to WSQ2.

We were so impressed with the performance of the FSQ modem under NVIS conditions, and also with how the 'peak hits' decoder worked at the higher baud rates, that we decided to introduce selective calling then and there, before anything at all was released.

My aim in this was to provide a really reliable and simple system for emergency communications. In an emergency situation, local and long distance communications are well served by (a) simplex VHF and (b) landline or satellite phone respectively, but the intermediate distances (typically 10 to 300 km) are not easy to cover without infrastructure (repeaters, Internet, cellular phone). This was made very clear after the Christchurch earthquakes, as communications even across the city proved impossible.

NVIS ionospheric propagation, with all its attendant technical problems, is all that remains viable over the 10 to 300 km range when the radio transmission towers are down, satellite dishes overturned[44], and when there are no phones and no power.

So FSQ, with its excellent NVIS performance, was seen as the answer, provided it was tailored for Emergency Communications use. I felt this would need to include selective calling. I saw FSQCall both as a simple means of directing field stations (link establishment and frequency management), and also to provide an 'order wire' function, where longer messages can be coordinated via FSQCall for transmission on another mode or frequency as necessary. The military ALE system is frequently used in this way.

[44] This happened in Vanuatu during Cyclone Pam in March 2015 – and they had no HF radio backup!

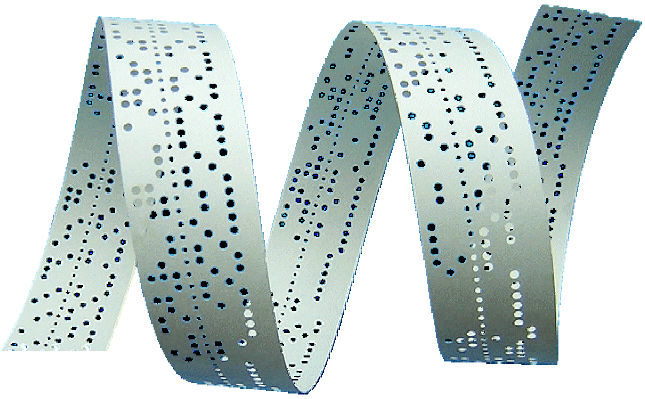
Registration, Multi-frequency and Multi-operator use

But in addition, I wanted FSQCall to be completely *ad hoc*. Stations should not be required to provide any pre-registration, should not need any radios, channels and IDs pre-programmed; essentially avoiding all the associated set-up business that pertains with systems such as ALE. I also wanted the system to be simple to use, require a minimum of training, and also be *fun to use* – this is a strategy to encourage operator training and experience without them realising it is happening!

Commercial and military Automatic Link Establishment systems use multiple frequencies, which are routinely scanned to find the best path when a message needs to be sent. I felt that the added complication of scanning radios, wideband antennae and the extra training required was unnecessary, and so opted for a single-frequency design. It's not difficult to teach users to switch to 3580 kHz at sunset and back to 7105 kHz at sunrise, for example. A base station can keep the field stations under the thumb by simply operating two radios, one on each frequency.

FSQCall has also been designed so that a multi-operator station (say at a larger base) can have separate receive-only operators feeding messages via a local network so they can be transmitted automatically via the main transmitting station, or a particular transmitter. You can also send images and files from other computers on a network.

The message logging system in FSQCall records every message received, which is a way of ensuring nothing is missed. It also allows *post mortems* to be carried out after an exercise, in order to determine where any problems might lie.



16. FSQCall

I'll describe this mode in some detail, since its development was quite involved, took a lot of effort and testing, and includes a host of new ideas. It might help you appreciate how every aspect of modern digital mode design is interconnected with everything else.

FSQCall contains the world's first open protocol *ad hoc* selective calling and link establishment and management system. Everything is fully disclosed, including the sentence structure, protocol and the program source code.

First of all I spent a few weeks working on a suitable protocol that would do what was needed for a simple yet versatile system, and at the same time was open-ended enough that further features and functions could be added later. The idea was based on using a free-form callsign as the station ID in all cases. It is also possible to use functional names ('base', 'townhall', 'car3' etc) instead of callsigns if that helps for exercises and emergency operation. I wrote a formal specification, which later morphed into the Syntax and Rules documents that accompany the software.

The first experimental versions of FSQCall introduced the protocol with the header containing the source address, checksum, addressee and command. This hasn't changed at all since. There were commands provided for sending selective text, a query for signal report (commonly known now as a 'ping'), speed change commands, even a message store command.

I think the first release made was V0.08, in January 2015. It caused quite a stir at first, mostly because some couldn't understand the sentence mode, and didn't take the time to read the help information! This version defaulted to non-Selcall mode, and when in Selcall mode, an extra Monitor pane appeared, tabbed with other small panes, a Heard List, and Syntax and Operating Rules. These latter two tabs reinforced the importance of understanding how the program operated.

By the time Version 0.24 was released, (early 2015) there was an FSQCall User Group,[45] and wide recognition that FSQCall really did perform well, despite, even perhaps because, there was no error correction. Version 0.24 had the file transfer and image transfer modes for the first time. In April 2015 professional programmers Bob NW8L and Mike KA4CDN released a 'US Version' of FSQCall. They felt it needed slightly different commands, and they added a couple of extra ones.

Because their fork of the program was relatively early, and has not been updated since, the US version lacks some of the later developments. Later in 2015, Dave W1HKJ (another professional programmer) added FSQCall to his FLDIGI suite. I can imagine it was quite a difficult marriage, as FLDIGI had no modes running in sentence mode at the time. Even today the FLDIGI interface for FSQCall operation is rather different. FLDIGI has images modes, but not all are the same as those in FSQCall. But it works well, and has certainly helped popularise the mode. Importantly, FLDIGI has allowed Linux and MAC systems to use the mode.

Publicity

By the way, in case you were wondering how the world gets to hear about each new development we made, it has been a determined policy. I have been in charge of this aspect

[45] https://groups.yahoo.com/neo/groups/FSQdigital/info

since about 2006. I use several techniques. Of course I first build a release package (software, help files and installation information), which I place on my web site.[46] That is accompanied as necessary with a new or updated web page with links to the new mode. No small job, I assure you!

The next step is to tell the world about the new development. The fastest way to do this is to write a press release, which is sent to interested mail groups, to on-line magazines, my radio friends mailing list, and the editors of print magazines such as AR (Australia), QST (USA) and Radcomm (RSGB).

I have also, when possible, written papers for International Conferences, such as the annual ARRL/TAPR sponsored Conferences in the USA.[47]

The final step, for anything really momentous, is to write and publish an article in QST, the most widely read ham magazine in the world, as was done for MFSK16 (2000), FSQCall (2015) and WSQCall (2018). As it happens, the first two of these also won 'Best Article' awards. The trouble with magazine articles is the publishing delay, so you have to write well in advance, at the risk of the articles being out of date by the time they are in print, given the way software has a tendency toward continuous improvement!

Image Modes

Con is an expert in handling image modes, and I know that he relished developing the FSQCall image transfer features. You can send images from file, from copy-and-paste, and from a web cam. You can also remotely request images from file and web cam, although for privacy and security reasons these are restricted (files from a Shared folder, web cam only when enabled). Like the text and file transmission capability, images can be transmitted to and received from selected recipients. They can also be sent to every station on the channel.

We developed four image modes. They operate in much the same way as SSTV, using FM modulation, but the bandwidth is restricted to that of FSQCall (about 300 Hz). The main difference is that there is no synchronising information sent with the picture. The picture size is sent ahead with the start command, and the receiver simply records *x* rows of *y* pixels in a synchronous manner. This is very like the image mode introduced to MFSK16 back in 2000.

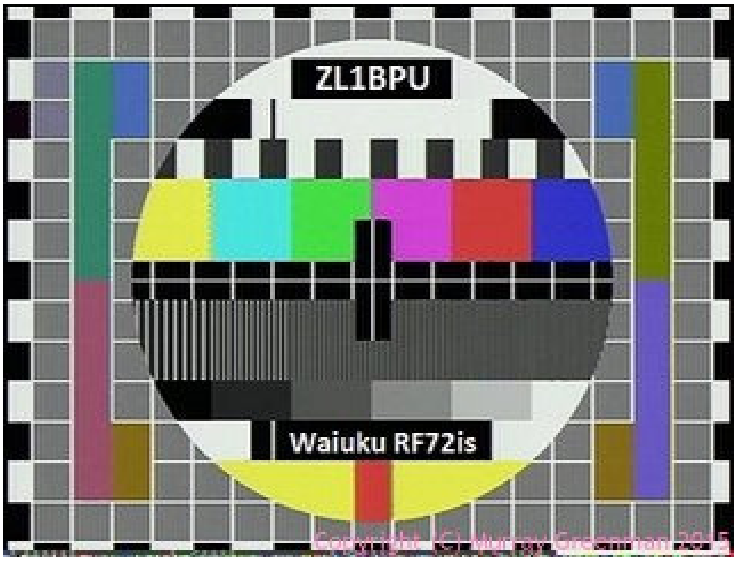


Fig. 62. A 640 x 480 pixel image, as received

[46] http://www.qsl.net/zl1bpu/

[47] *An MFSK Mode for HF DX*, Proc 19th ARRL & TAPR DCC, ARRL, 2000. ISBN 0-87259-814-4

The four image modes are single frame, 4:3 format. The low-resolution colour mode is 160 x 120 pixels and takes 48 seconds to transmit. The high-resolution colour mode is 320 x 240 pixels, and takes 192 seconds to transmit. The HD colour mode is 640 x 480 pixels, with duration of 256 seconds.

There is also a B&W HD mode, 640 x 480 pixels, which takes 256 seconds to transmit.

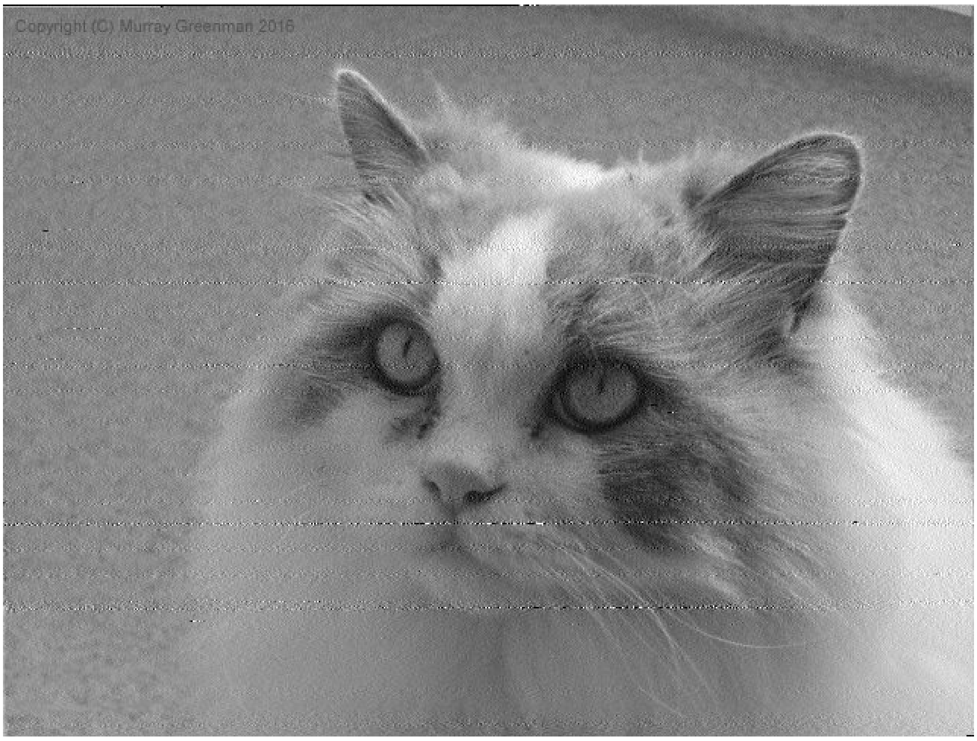


Fig. 63. B&W 640 x 480 image

The remarkable quality of this image (Figure 63) is made even more impressive when I reveal that this picture is actually the result of two successive transmissions, each over 330 km on 80m at night!

The B&W mode is also an excellent way to send hand-written messages or standard emergency message forms. These can be captured with a scanner, a web cam or (at a pinch) a phone or tablet.

Black and white drawings, maps and schematics are also excellent candidates for the HD B&W mode. Here's an example, also transmitted twice on 80m at night:

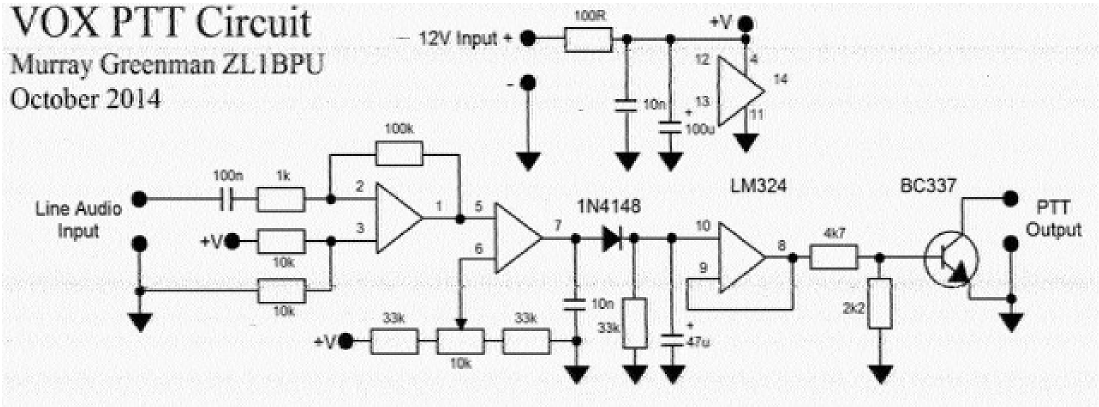


Fig. 64. A line drawing sent via FSQCall

The one thing about the image modes, which for me remains a disappointment, is that there is no accurate timing of the start of reception relative to the start of transmission. As a result, although picture reception starts automatically, the pictures need to be manually aligned after reception, and in the case of colour pictures, this means choosing the correct frame order to give the correct colours.

The MFSK16 image mode, although simpler, did achieve automatic synchronism by using a fixed start delay. I'd still like to see something like this in FSQCall.

Functions and Commands

FSQCall can send plain text, commands, images (with automatic reception) and files. You can also append to existing files. You can also query a list of the available images or files, and use the information to request automatic transmission of these images and files.

The list of available one – character commands is quite versatile:

?	Query SNR
$	Query Heard List
@	Query location
&	Query station message
^	Query software version
_	Query squelch setting
<	Reduce transmitting speed
>	Increase transmitting speed
*	Activate FSQCall mode
#	Send file
+	Request file
|	Send alert
!	Relay message
~	Delayed relay message
%	Send/receive image

The one-character commands are called 'triggers', and are readily learned. To send a simple text message to the other station, the message trigger used is simply a 'space', for example:

zl3xyz are you busy?

Lower case is preferred where possible, as it is not only twice as fast to transmit as upper case; it also has half the error rate. The callsign recognition system is case sensitive, so ZL3XYZ is a completely different station to zl3xyz.

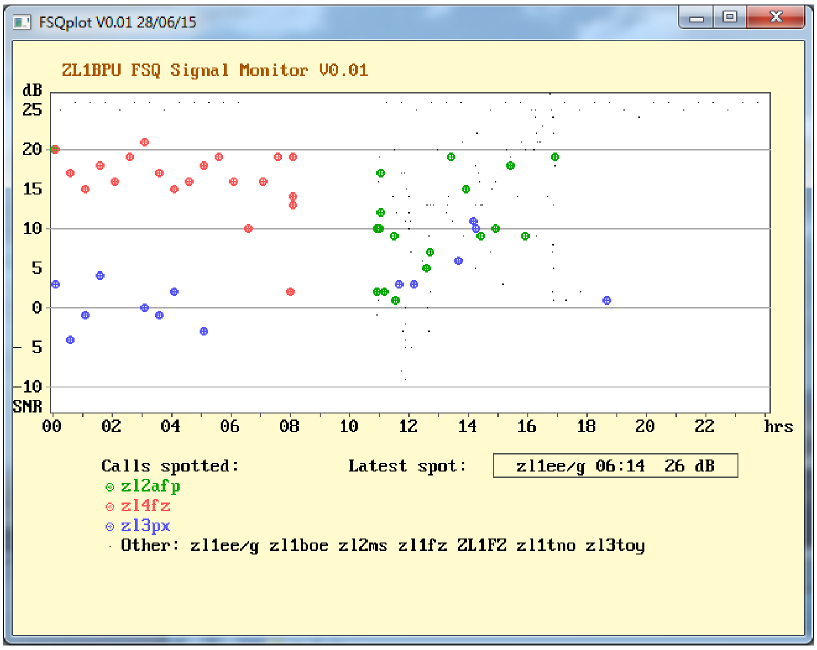
17. FSQCall Bells and Whistles

FSQCall was designed to communicate with other ‘helper’ programs by way of file interchange. Several of the ‘bells and whistles’ (enhancements) operate this way, which encourages development by third party programmers.

FSQplot

The first of these helpers is something I wrote. FSQplot reads the Heardlog file, and plots the stations in it as a graph of Signal to Noise Ratio (SNR) against time. The time scale is 24 hour, and so not only do you get a very good idea of propagation over the course of the day, but also you can plot for days on end, and the plots are superimposed.

Another benefit is that you can clearly see what times of day each station is active, or is within propagation range. If you study a single station over several days, you can build up a very clear image of how propagation varies on the mutual path.



Calls spotted:

Latest spot: zl1ee/g 06:14 26 dB

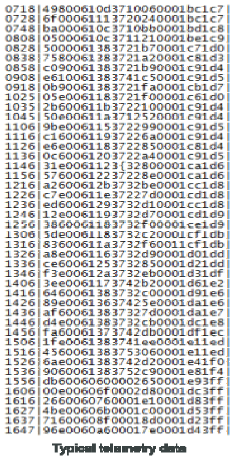
⊗ zl2afp
⊗ zl4fz
⊗ zl3px
· Other: zl1ee/g zl1boe zl2ms zl1fz ZL1FZ zl1tno zl3toy

Fig. 65. FSQPlot shows Heard Log entries graphically

Telemetry

FSQCall is also able to read a special file ‘data.txt’ placed in the /Shared folder. When it discovers a file there, it dumps the contents into the keyboard buffer, and transmits it! So any other application (or another user on a network) that has a file to send, can easily do so, provided they place the appropriate preamble at the start of the file.

The first application for this technique was a telemetry program I wrote. This can collect data for a given time (say an hour) and send all the data once each hour, or can send each individual frame as soon as data is available.



```
0718|49800610d3710060001bc1c7|
0728|6f0006111372024000 1bc1c7|
0748|ba000610c3710bb0001bd1c8|
0808|05000610c37112100 01be1c9|
0828|5000061383721b70001c71d0|
0838|7580061383721a20001c81d3|
0858|c090061383721b90001c91d4|
0908|e610061383741c50001c91d5|
0918|0b90061383721fa0001cb1d7|
1025|05e0061183721f00001c61d0|
1035|2b600611b37221000 01c91d4|
1045|50e00611a37125200 01c91d4|
1106|9be00611537229900 01c91d5|
1116|c160061193722 6a0001c91d4|
1126|e6e0061183722850001c81d4|
1136|0c60061203722a40001c91d5|
1146|31e006112 3{3280000 1ca1d6|
1156|5760061223722 8e0001ca1d6|
1216|a260061 2b3732be0001cc1d8|
1226|c7e00611e37227d0001cd1d8|
1236|ed60061293732d10001cc1d8|
1246|12e00611937 32d70001cd1d9|
1256|3860061183732f00001ce1d9|
1306|5de0061183732c20001cf1db|
1316|83600611a3732f60011cf1db|
1326|a8e0061163732d90001d01dd|
1336|ce60061253732850001d21dd|
1346|f3e00612a3732eb0001d31df|
1406|3ee006117 3742b20001d61e2|
1416|6460061383732c00001d91e6|
1426|89e00613637425e0001da1e6|
1436|af600613837327d0001da1e7|
1446|d4e0061383732cb0001dc1e8|
1456|fa60061373742db0001df1ec|
1506|1fe00613837 41ee0001e11ed|
1516|4560061383753060001e11ed|
1526|6ae0061383742d20001e41f0|
1536|9060061383752c90001e81f4|
1556|db60060600002650001e93ff|
1606|00e00606f0002d80001dc3ff|
1616|266006076000 1e10001d83ff|
1627|4be00606b0001c00001d53ff|
1637|71600608f00018d0001d23ff|
1647|96e0060a600017e0001d43ff|
```

Typical telemetry data

Fig. 66. Received telemetry data

The file store function (#) places files in the /Shared folder, and you can remotely define a file name and place data from multiple transmissions appended into it, by simply addressing the file appropriately.

By setting the address preamble and a file name in telemetry program (FST.exe, Fast Simple Telemetry), the design allows the telemetry to be delivered to one or more stations, and the data would be appended to the specific file at the recipient computer. In this way, multiple remote telemetry stations could deliver their data to their own file at a chosen recipient.

The telemetry device I developed used an Arduino Uno with an analogue input 'shield' which I designed, and I used it to collect ambient and transmitter temperature, battery voltage and charge/discharge currents, from the remote HF station it was connected to. This device communicates with FST (written in Just BASIC) via USB, and FST then saves the file for FSQCall (on the same computer in this case) to transmit. Figure 65 shows several hours of telemetry data received at my home station from this set-up.

I also wrote a graphical application to show these data on the client computer. Figure 67 is a block diagram of how this telemetry system works, although there are many variations.

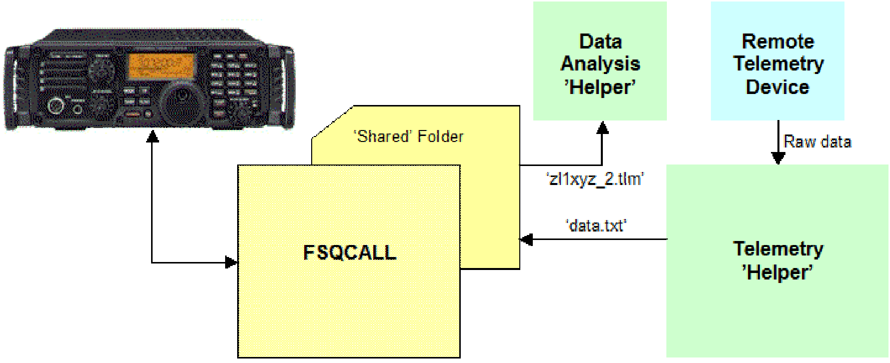


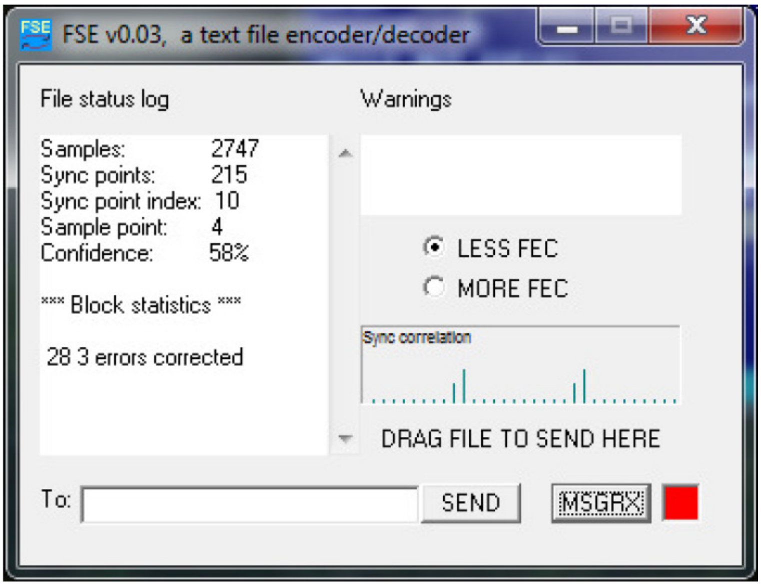
Fig. 67. One example of a telemetry reporting arrangement

Error Correction

Under most circumstances, FSQCall provides near perfect reception, and the same applies to sending files. However, there are a few circumstances, especially in Emergency Communications, where even trivial errors cannot be tolerated, and the receiving operator needs to know not only that the file received is correct, but is *guaranteed* correct. For example, if lists of casualty names addresses or phone numbers or even medical details are involved.

No error correction is applied to normal FSQCall transactions, just to file transmissions.

The Helper program FSE (Fast Simple Error Correction) can be used to send files that will almost always be received 100% correctly, and if not, the recipient will be informed of the presence of an error. FSE uses off-line Reed-Solomon error encoding to generate a text file that is sent the same as any other file, and is then read and corrected by the same program at the receiver.



```
Samples:          2747
Sync points:      215
Sync point index: 10
Sample point:     4
Confidence:       58%

*** Block statistics ***

 28 3 errors corrected
```

● LESS FEC

○ MORE FEC

Fig. 68. The FSE error correction helper program.

But the functionality of this helper program is much more complex than immediately appears. Any digital transmission is subject to random errors, for example wrong symbols, missed symbols, or extra symbols, which is why FSE exists in the first place, but MFSK modes, especially ones using a varicoded alphabet, are especially prone to two types of errors which no error correction system can easily correct: *missing* symbols or *extra* symbols.

To counter that, FSE uses a pre-coding system that sends all characters as single symbols (so no symbol is ever dependent on another), along with a synchronous decoding scheme that can *never* have missing or extra symbols, just incorrect ones.

This synchronous business was another first for Amateur radio – the use of a synchronous FFT decoder and a synchronising system based on cross-correlation. This complex business is beyond what can be described here, but is well documented.[48]

A special file containing raw FFT sample data is written by FSQCall during reception, 12 samples per symbol, and this is sampled by FSE to determine symbol sync, and then to synchronously decode the message so that it can then be run through the FEC correction with no missing or extra symbols.

FSE was bundled with FSQCall from 2017, but has not seen a great deal of use, partly because there has not yet been the need for it, but also it is more straightforward to simply send a document as a high resolution B&W image (like a FAX), which can be noisy, but can't easily contain undetected errors.

The concept of the synchronous decoder and a cross-correlator were later reused for enhanced full-time reception in WSQCall (see later).

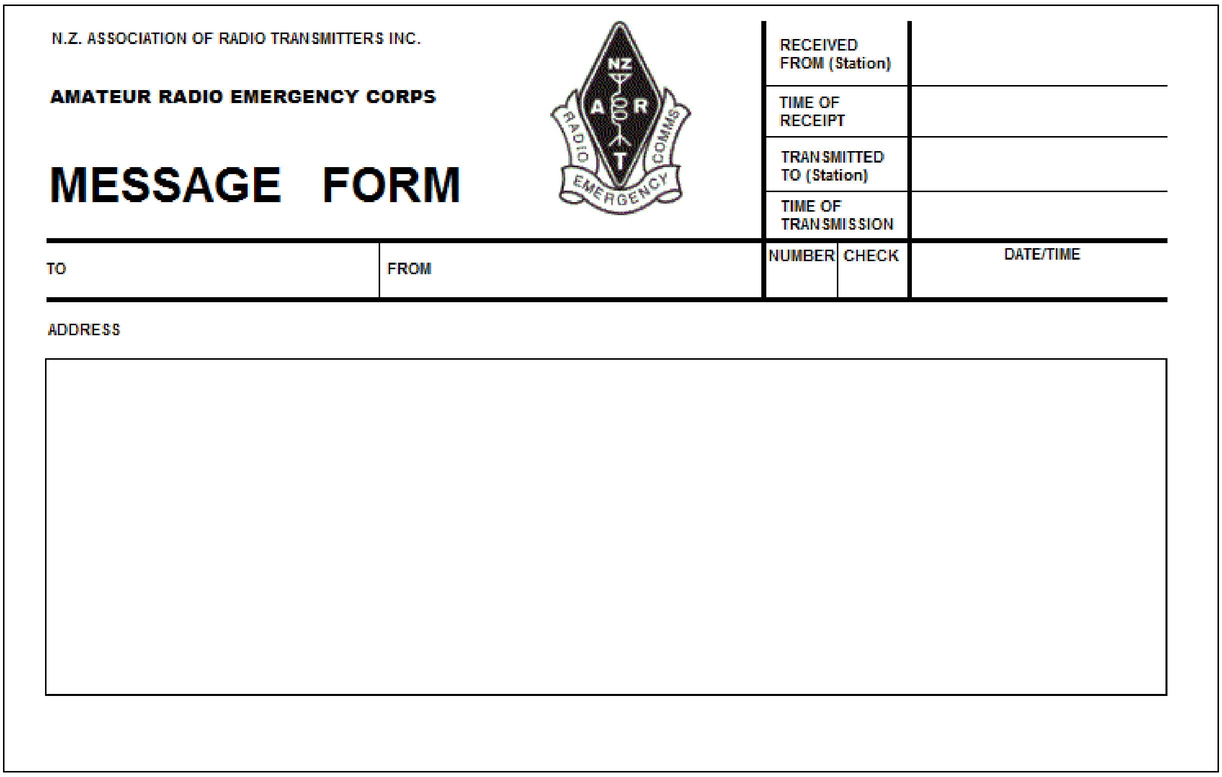
[48] http://www.qsl.net/zl1bpu/DOCS/FSQ%20and%20Error%20Correction.pdf

Forms and Form Designer

We developed a way to efficiently send standard messages as forms, where the sender filled out fields on the form, but only field number and the data was sent. The recipient would already have the appropriate form, and the FSM (Fast Simple Messaging) helper would present the data using the form, which could then be printed.

The system was quick and effective, but there were problems, especially with complex forms, as in addition to a bit map of the form being required, the form required a definition file indicating where each field was. We never successfully made a reliable form designer, and the few successful forms we used required painstaking manual definition. Complex forms proved too difficult.

Again, it proved simpler to send the message as a FAX. It took slightly longer, but was simpler to do and just as reliable – and not dependent on having a previously defined form.



N.Z. ASSOCIATION OF RADIO TRANSMITTERS INC.

AMATEUR RADIO EMERGENCY CORPS

MESSAGE FORM

RECEIVED FROM (Station)	
TIME OF RECEIPT	
TRANSMITTED TO (Station)	
TIME OF TRANSMISSION	

TO	FROM	NUMBER	CHECK	DATE/TIME

ADDRESS

Fig. 69. A typical message form

Fast Simple Privacy

I wrote this application, FSPlite, in QB64, back in 2015. It is essentially a stand-alone substitution code application, and could be used for applications other than FSQCall, since the on-air format consists of simple text files, albeit unreadable. It accepts a source file, and encrypts it. Then you send the encrypted file. The same program will decrypt the message, which is impossible without knowing the encryption key.

Substitution codes have a poor reputation, since they are more readily cracked than other types. In this situation however, the 1:1 correspondence between source and encrypted characters has an important function: any character error in the transmitted text results in just one corresponding error in the plain text. The error is not propagated further and does not completely prevent decryption, as would happen with more complex schemes, such as the rolling code types.

The coding system is quite simple. As you start the program, it generates 9999 random code alphabets in memory, with 1:1 letter substitution. You choose one of these by setting a PIN number. Since the algorithm for the alphabet generation is not available outside the program, decrypting the messages by brute force would take longer than the time frame for which the stolen message would be useful.

There are three remarkable features to what I developed:

(1) The encrypted file is the same size as the source (thanks to substitution coding);
(2) The system isn't prone to insertions and deletions, as all characters in the encrypted file are further encoded to be single symbol (and again the encoded file is the same size!
(3) A 'PIN Number' system is used for encryption keys so they can be changed hourly or daily. There's even a special 'Test PIN' (0000), which encodes and decodes for training purposes without encryption. This can be used for training with amateur callsigns, since it's still more-or-less human readable.

FSP hasn't been used a great deal, but it has created enormous interest. It cannot of course be used for Amateur transmissions, where encryption is not permitted. It was developed to prevent the press and voyeurs from reading sensitive emergency traffic. This has happened in the past, as emergency staff has been asked targeted questions by the press, who already knew some of the details by reading radio traffic.

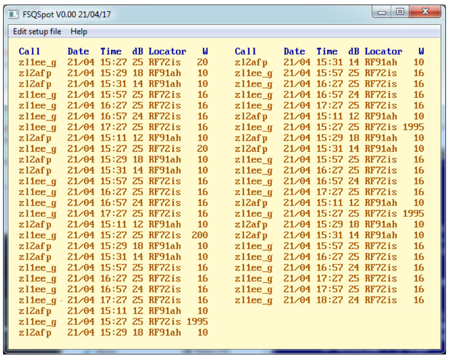
Logging

As mentioned, FSQCall keeps a log of all stations heard, along with the time and their SNR. The file is in chronological order. Importantly, it contains only stations whose callsign is verified against the accompanying checksum.

There is also a Message Log, similar in concept, but in addition it appends the complete payload of every message. This is useful if you need to go back and review a conversation, or after a message-passing exercise, in order to piece together what went wrong.

There is also a Locator log, similar in concept to the Message Log, except that it saves only received messages with a correctly formatted locator, which matches its checksum. FSQCall can have its Sounding feature modified to send Locator sentences. As yet there is no way to share these sentences on the Internet or analyse them, although the concept is intended to mirror the WSPR capability in this regard.

Of course third party applications can make use of all these log files. I wrote a simple program in 2017 to analyse and display the data from the station Locator Log. I called the program FSQSpot.



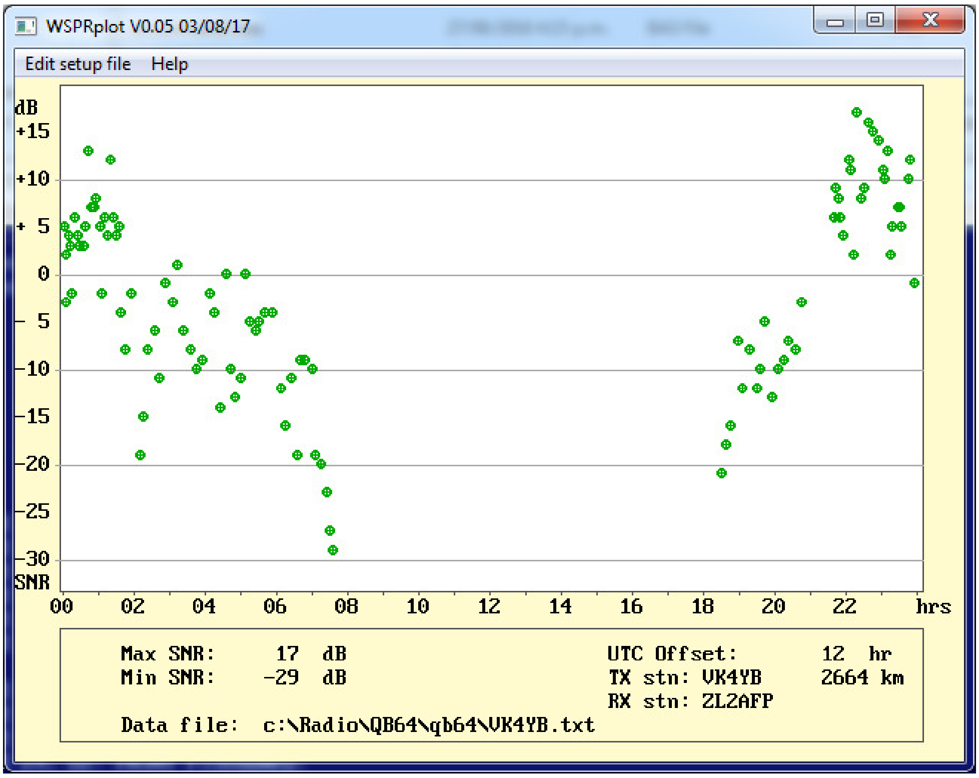
```
Call     Date  Time  dB Locator   W     Call     Date  Time  dB Locator   W
zl1ee_g  21/04 15:27 25 RF72is   20     zl2afp   21/04 15:31 14 RF91ah   10
zl2afp   21/04 15:29 18 RF91ah   10     zl1ee_g  21/04 15:57 25 RF72is   16
zl2afp   21/04 15:31 14 RF91ah   10     zl1ee_g  21/04 16:27 25 RF72is   16
zl1ee_g  21/04 15:57 25 RF72is   16     zl1ee_g  21/04 16:57 24 RF72is   16
zl1ee_g  21/04 16:27 25 RF72is   16     zl1ee_g  21/04 17:27 25 RF72is   16
zl1ee_g  21/04 16:57 24 RF72is   16     zl2afp   21/04 15:11 12 RF91ah   10
zl1ee_g  21/04 17:27 25 RF72is   16     zl1ee_g  21/04 15:27 25 RF72is 1995
zl2afp   21/04 15:11 12 RF91ah   10     zl2afp   21/04 15:29 18 RF91ah   10
zl1ee_g  21/04 15:27 25 RF72is   20     zl2afp   21/04 15:31 14 RF91ah   10
zl2afp   21/04 15:29 18 RF91ah   10     zl1ee_g  21/04 15:57 25 RF72is   16
zl2afp   21/04 15:31 14 RF91ah   10     zl1ee_g  21/04 16:27 25 RF72is   16
zl1ee_g  21/04 15:57 25 RF72is   16     zl1ee_g  21/04 16:57 24 RF72is   16
zl1ee_g  21/04 16:27 25 RF72is   16     zl1ee_g  21/04 17:27 25 RF72is   16
zl1ee_g  21/04 16:57 24 RF72is   16     zl2afp   21/04 15:11 12 RF91ah   10
zl1ee_g  21/04 17:27 25 RF72is   16     zl1ee_g  21/04 15:27 25 RF72is 1995
zl2afp   21/04 15:11 12 RF91ah   10     zl2afp   21/04 15:29 18 RF91ah   10
zl1ee_g  21/04 15:27 25 RF72is  200     zl2afp   21/04 15:31 14 RF91ah   10
zl2afp   21/04 15:29 18 RF91ah   10     zl1ee_g  21/04 15:57 25 RF72is   16
zl2afp   21/04 15:31 14 RF91ah   10     zl1ee_g  21/04 16:27 25 RF72is   16
zl1ee_g  21/04 15:57 25 RF72is   16     zl1ee_g  21/04 16:57 24 RF72is   16
zl1ee_g  21/04 16:27 25 RF72is   16     zl1ee_g  21/04 17:27 25 RF72is   16
zl1ee_g  21/04 16:57 24 RF72is   16     zl1ee_g  21/04 17:57 25 RF72is   16
zl1ee_g  21/04 17:27 25 RF72is   16     zl1ee_g  21/04 18:27 24 RF72is   16
zl2afp   21/04 15:11 12 RF91ah   10
zl1ee_g  21/04 15:27 25 RF72is 1995
zl2afp   21/04 15:29 18 RF91ah   10
```

Fig. 70. An FSQSpot listing.

18. WSPRplot

In 2017 I wrote a simple program that looked rather like FSQplot, and had a similar function, plotting propagation of a station over 24 hours or even several days.

The difference is that with WSPRplot, the source data is not a local file, but the display from the online WSPR database, which you simply capture as text and save as a file.



```
Max SNR:    17  dB                           UTC Offset:      12  hr
Min SNR:   -29  dB                           TX stn: VK4YB    2664 km
                                             RX stn: ZL2AFP
Data file:  c:\Radio\QB64\qb64\VK4YB.txt
```

Fig. 71. WSPRplot of VK4YB running WSPR on 474 kHz, as received by ZL2AFP

The remarkable thing about this simple tool is that you can use it to analyse the path between any two stations in the world, for any period! In the above example, I used data reported by ZL2AFP of his reception of VK4YB on 630 metres. This is achieved by first setting up the appropriate filters on the WSPRnet web site, then capturing and plotting the data.

Parsing the data proved to be very tricky, as the web information has hidden non-printing characters such as tabs. In addition while I was developing the software, they changed the display format! I also discovered that what you capture depends on the web browser you use.

You can see in Figure 70 that propagation drops off quite abruptly at sunrise (0730 local time) and picks up again at sunset, 1830. You can also see that signal to noise ratio reports varied by over 40 dB.

The WSPR database always reports time in UTC. The program allows for any local time offset, and as you can see in the example, I have set +12 hours offset so it displays local time, with daytime in the middle.

19. WSQCall

By now you will have seen how each development of a new mode frequently evolves from prior work, and especially in the case of FSQCall, is strongly influenced by discoveries made during the WSQ2 effort.

FSQCall V0.42 (the latest system) is an impressive program, with a comprehensive suite of features and add-ons. It was perhaps inevitable that we should next look back at WSQ2, and think about how it could be improved.

So of course the next version of WSQ would also offer a selective calling protocol.

Modem Improvements

The discoveries made around the advantage of a 3/T tone spacing (and receiver FFT with 3 bins per tone) led us to test 3/T in a WSQ type program. We also wanted to make a multi-speed version, so we could experiment with lower symbol rates without compromising the 3/T arrangement.[49]

The complete receiver code section in these modes is clocked from the sound card clock, but in the new WSQCall program, changing speed was achieved differently. In FSQCall, the wide-range 'peak hits' decoder was used, and a fixed FFT was used for all speeds received. For WSQCall, a decision was made to match the decoder optimally to the symbol rate and still use a single FFT, with speed changed by decimation. This also meant using different tone spacings at different speeds. The reason for this will be apparent later.

The new receiver was achieved using just one set of receiver code, but involved decimating the sound card data (and thus the overall rate at which the FFT worked). Of course this made the mode no longer compatible with WSQ2, but allowed any number of different speeds and tone spacings (we chose three), simply by changing the decimation ratio, each speed thus having the same optimum tone spacing and bin spacing arrangement. Furthermore, each symbol would now be found in 16 consecutive FFT solutions, i.e. it could be synchronous with the FFT.

When the FFT operating speed is changed without changing its size, the bin spacing changes correspondingly. Since this was now happening through decimation, we needed to change the transmitted tones not only in symbol rate, but also in tone spacing, to maintain the 3/T ratio.

There were other small improvements made to the peak-hits decoder (offering *three cumulative hits* as an alternative to *three consecutive hits*), and improvements to the FFT, which reduced the effect of noise. But the main advance in the new WSQ effort was the business of making everything synchronous with the sound card clock.

Synchronous Reception

You will recall that in FSQCall we offered an FEC file transfer 'helper', which used a synchronous decoder, in order to achieve accurate error correction and avoid insertion and deletion errors. This was made possible by the FSQCall program writing a file (raw.RS) containing all the samples from the FFT process as received. In FSQCall this amounted to 12 samples per symbol. The helper then analysed the file using a synchronous decoder.

[49] In FSQCall, the relationship between tone spacing and symbol rate varies with symbol rate, while the receiver FFT spacing is fixed.

This capability was retained in WSQ, but the sample rate with the new decimation business was now always 16 samples per symbol. While Con worked on other aspects of the program, I got to work devising a fully synchronous decoder, based on a cross-correlator, to discover symbol sync, and then synchronously sampling the raw.RS file once reception was over.

To achieve this, I first discovered the ‘change points’ (where one symbol changed to another) in the sample file, which with an IFK+ modulation scheme is easily achieved by subtracting one sample from the next and squaring the difference. Successive samples of the same tone would return a value of zero, or at the most perhaps 2 if there were slight drift. However, when the symbol changed, the value would be much larger.

My program operated a really simple cross correlator. By knowing that the symbols were always 16 samples long, I could operate a 16-cell array, which was cleared before I examined the file. It used a pointer (0 – 15) incremented at each sample through the file, returning to zero at every 16th sample. When the sample indicated a change point, the corresponding cell in the array was incremented. So after the whole file had been examined, it would be obvious where the change points were by examining the array. You can display the array contents graphically:

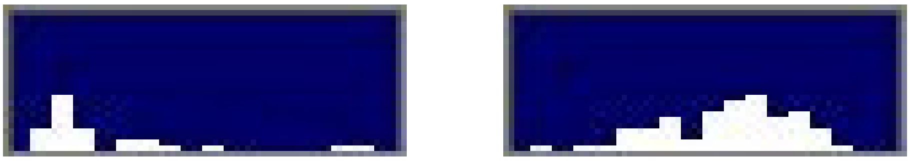


Fig. 72. Cross-correlation array, strong signal (left), weak signal (right)

It’s clear in these pictures (real reception from WSQCall), even with weak signals, where most of the change points occurred in each case.

The purpose of exploring synchronous decoding was that we knew, especially when signals are weak, that the tones become quite distorted, particularly at the edges, and we wanted to ensure we could sample the data in the relatively clean centre of each symbol. Figure 73 shows an example from real data.

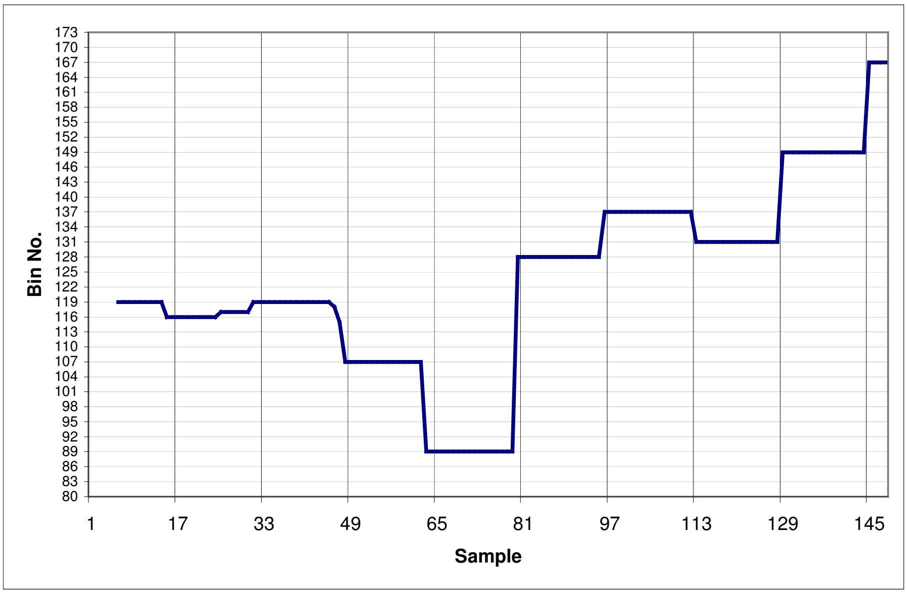


Fig. 73. Real WSQ data, displayed as FFT bin number vs. sample number.

You can see even in this small sample (the signal wasn't even a weak one) how the samples aren't always every three bins apart (as they should be with 3/T), and worse still, the timing does not coincide accurately with the sampling. This can lead to errors in the peak-hits decoder. Figure 74 shows the same data as Figure 73, but rearranged with two nominal change points, superimposed as an 'eye diagram'. It should be easy to see how making samples of the data near these points could lead to errors. The best place to sample is right between the two vertical lines, the centre of the eye diagram.

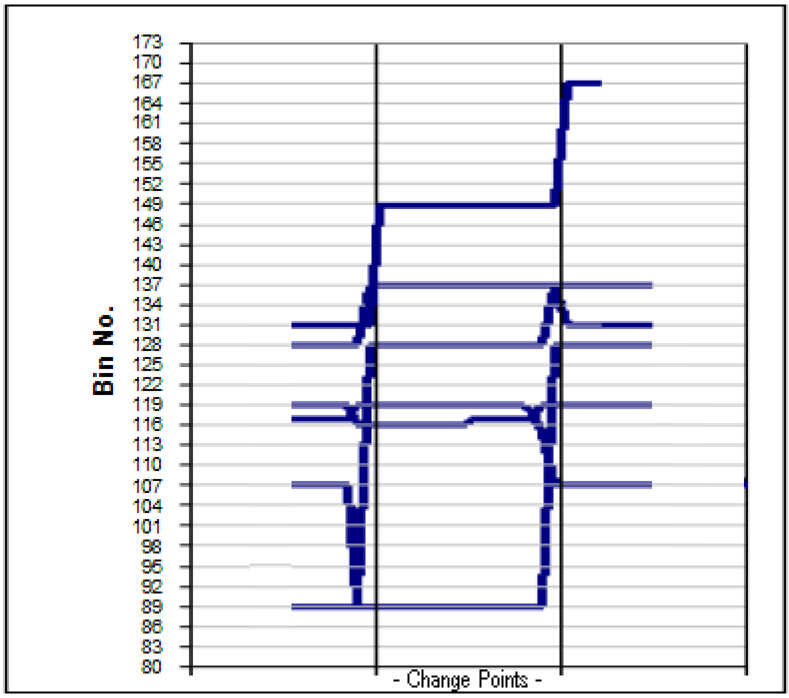


Fig. 74. Eye diagram of the data in Fig. 73.

I can illustrate this problem in another way, using data from the program for the same two signals as shown in Figure 72.

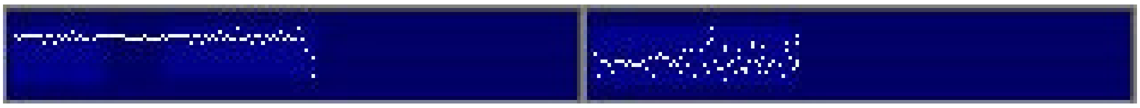


Fig. 75. Sync history, strong signal (left), weak signal (right)

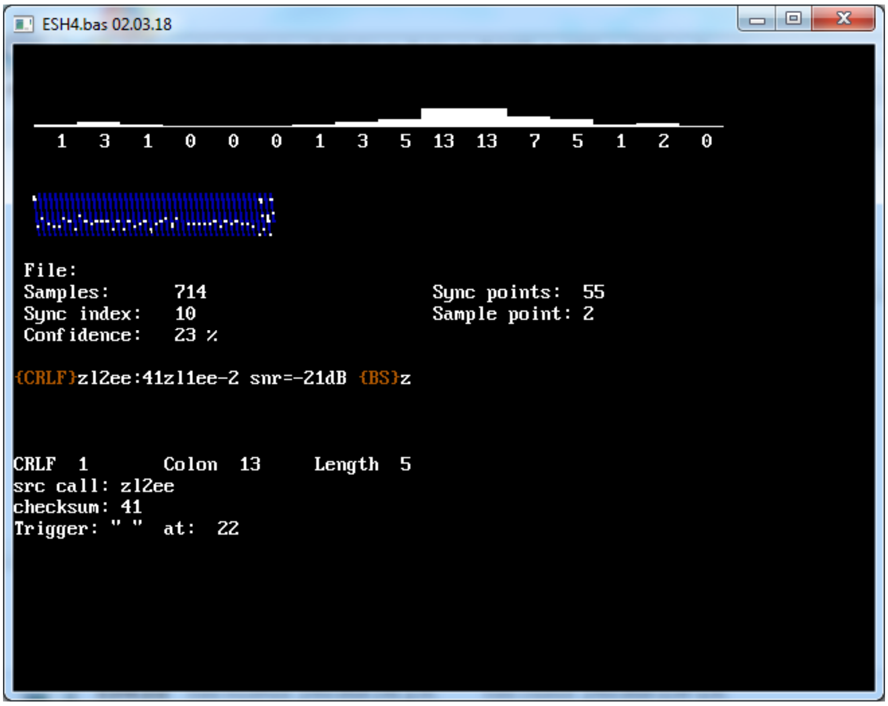
In these graphs, Figure 75, also displayed by the program, the graphs are 16 FFT samples high, and as long as the duration of each transmission. Each dot represents a 'change point'. You can see that when the signal is strong, the change points are in a reasonably straight line, as they should be. When the signal is weak however, there is considerable variation. If you look closely, you may see that sometimes the dot is missing, or there is more than one dot per column, indicating how the 'peak hits' decoder could easily make a mistake.

If we were to sample the data as far as possible away from the change points, we could avoid these problems, and (so the thinking went), considerably improve accuracy of received messages.

The advantage of a cross-correlator, even a simple one, is that it is inherently immune to noise. The more samples you have, the less affect any incorrect decisions (missing or extra change points) will have on the result.

Synchronous Decoding

Once we know where the change points (mostly) are, we can sample the file with confidence, knowing that we are sampling far away from where the change points mostly are. Since there are 16 samples per symbol, this will of course be 8 samples away from the correlator array peak. Figure 76 shows the user interface of my prototype synchronous decoder, having sampled a real transmission from the raw.RS file.



```
   1   3   1   0   0   0   1   3   5  13  13   7   5   1   2   0

 File:
 Samples:      714                    Sync points:  55
 Sync index:   10                     Sample point: 2
 Confidence:   23 %

{CRLF}zl2ee:41zl1ee-2 snr=-21dB {BS}z


CRLF  1       Colon  13     Length  5
src call: zl2ee
checksum: 41
Trigger: " "  at:  22
```

Fig. 76. The synchronous decoder prototype

The prototype program was written in QB64. At the top of the screen the correlator array is illustrated, bins 0 to 15, left to right. The numbers underneath are the total number of change points found at each index.

Below the correlator display is a ‘sync history’, the same idea as shown in Fig. 75. The bin with the most changes could be number 9 or 10, and looking at the sync history display in Figure 76, you can also see that most change point dots are around 9 or 10 places down from the top.

The next items in Figure 76 are statistical values from the file the program has decoded. 714 samples, 55 change points found, and the program ran a max-finding algorithm and chose index 10 as the place where most change points are. It then chose index 2 as the appropriate place to sample.

By the way, in 714 samples there should be 714/16 or approximately 44 change points. The program found 55, so was being affected by noise. Even so, the correlator sync point decision is still secure.

Below the statistical data in Figure 76 you see the complete decoded sentence that was received. Hidden (non-printing) characters are shown **{in brown}**.

Message Parser

The information at the bottom of the screen is that reported by a message parser, which is an algorithm that attempts to understand the message command. It located the source callsign, the checksum, the destination callsign, and the command trigger. This replicates what the FSQCall and WSQCall parsers needs to do with every arriving sentence.

```
{CRLF}zl2ee:41zl1ee-2 snr=-21dB {BS}
```

How does it do this? The source callsign is between the first **{CRLF}** and the colon. The checksum (for that callsign) is the two characters following the colon. The trigger character is then identified working onward from the left (in this case a space). The destination callsign is then everything between the checksum and the trigger character. The message payload is everything after the trigger up to the **{BS}** (backspace) end-of-message character.

It's a simple but versatile arrangement, and the FSQCall and WSQCall parser algorithms work in the same way.

If the checksum doesn't match, the message is rejected. The same applies if the destination callsign doesn't belong to the receiving station.

Implementing Synchronous Operation in WSQCall

I tested the decoder thoroughly over several weeks while Con was working on other aspects of the program, principally adding a cut-down message parser and command processor similar to FSQCall. He also made several improvements to the way callsigns are recognised.

I sent Con my synchronous decoder algorithm (in QB64 source code), and he rewrote it in ANSI C (quite difficult since the string and array handling in C is much more complicated than in QB64), and then added the synchronous decoder to the emerging WSQCall program. It worked right from the start, and gave good results. We had retained the peak-hits decoder, so it was interesting to see that when signals were weak, the synchronous decoder generally had the edge.

As a result, the functionality was swapped around: the synchronous decoder now displays output in the receive pane and also drives the parser and command processor, while the peak-hits decoder simply displays in the Monitor pane. We felt it was useful to retain the peak-hits decoder, partly to show what the differences are between decoders, but mostly because it displays in real time, as the tones arrive; the synchronous decoder is a post-processor, decoding only when the squelch has closed.

Subsequent tests showed that with synchronous decoding and the new FFT improvements, sensitivity improved slightly (to about –28 dB SNR), but more significantly, the intelligibility of weak signals, even at –29 dB SNR, was markedly improved.

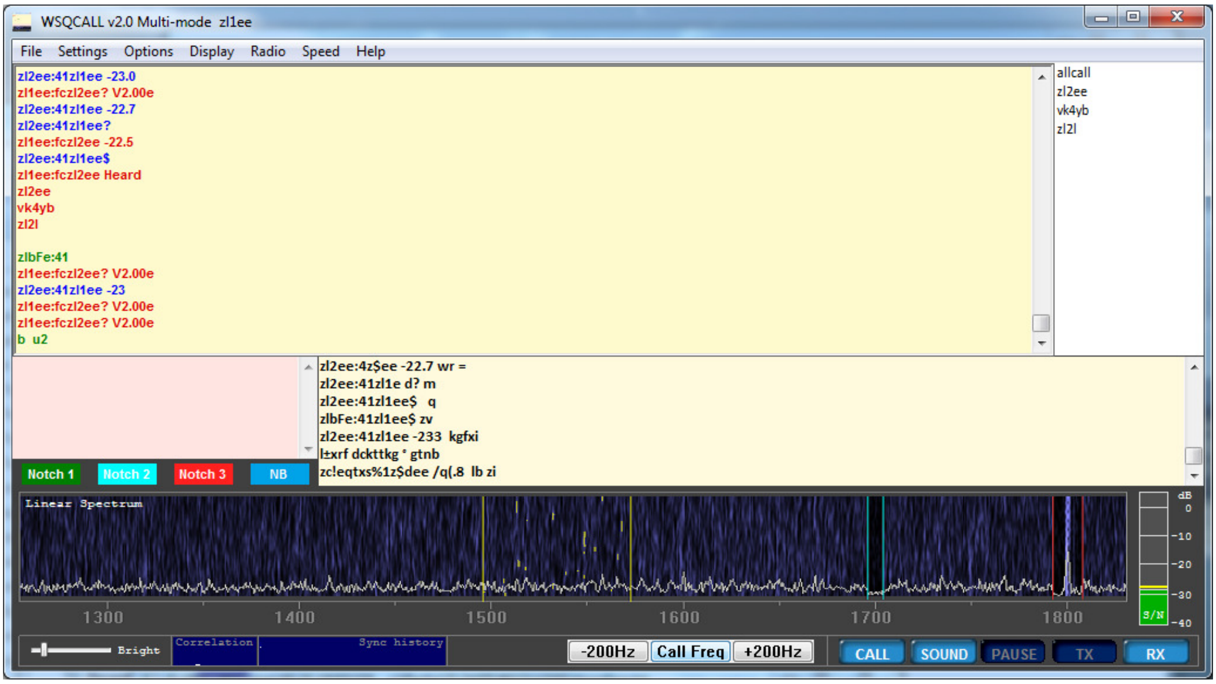
While WSQCall has no error correction, and thus no ability to ride through fades (as is the case with WSPR), we have demonstrated that you can copy signals with good intelligibility at similar signal levels. To achieve this performance without error correction has certainly not been done before!

Release

We released WSQCall at V1.1 in late January 2018, with the new modem but without the synchronous decoder. It included a mode compatible with WSQ2 and Wolf's WSQ2. In March 2018 V1.2 hit the streets, with the new synchronous decoder, and it has been a real hit.

WSQCall V1.2 includes the Heard '$' command, the '?', '@' and '&' information requests, the '!' relay command, and a new one, '+', 'repeat last message'. There are four notch filters and an effective noise blanker.

Con is currently working on WSQCall V2.0, which I am also testing. It includes a wider waterfall, rearranged panes, waterfall zoom and pan, plus ± 200 Hz QSY buttons.



```
zl2ee:41zl1ee -23.0
zl1ee:fczl2ee? V2.00e
zl2ee:41zl1ee -22.7
zl2ee:41zl1ee?
zl1ee:fczl2ee -22.5
zl2ee:41zl1ee$
zl1ee:fczl2ee Heard
zl2ee
vk4yb
zl2l

zlbFe:41
zl1ee:fczl2ee? V2.00e
zl2ee:41zl1ee -23
zl1ee:fczl2ee? V2.00e
zl1ee:fczl2ee? V2.00e
b  u2
```

```
zl2ee:4z$ee -22.7 wr =
zl2ee:41zl1e d? m
zl2ee:41zl1ee$   q
zlbFe:41zl1ee$ zv
zl2ee:41zl1ee -233  kgfxi
l±xrf dckttkg ' gtnb
zc!eqtxs%1z$dee /q(.8  lb zi
```

Fig. 77. The user interface for WSQCall V2.00e.

You can see along the bottom of the user interface the synchronous decoder 'metrics', the Correlation and the Sync History, as described earlier in this chapter.

Two notch filters have been deployed, as you can see on the waterfall display. The notches have adjustable width and depth. The noise blanker also has adjustable threshold. WSQCall V2.00 has not yet been released.

WSQCall also sports (from V1.20) a separate signal strength S-meter, shown as a small applet window.

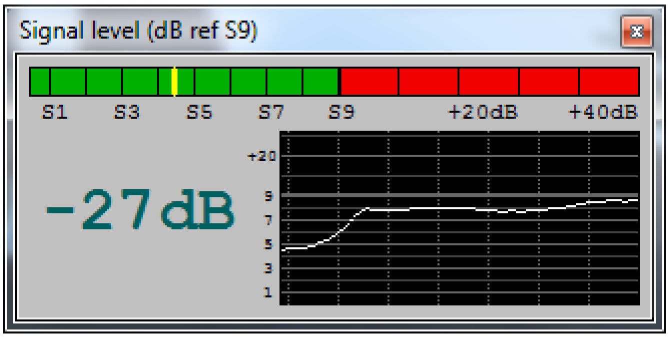
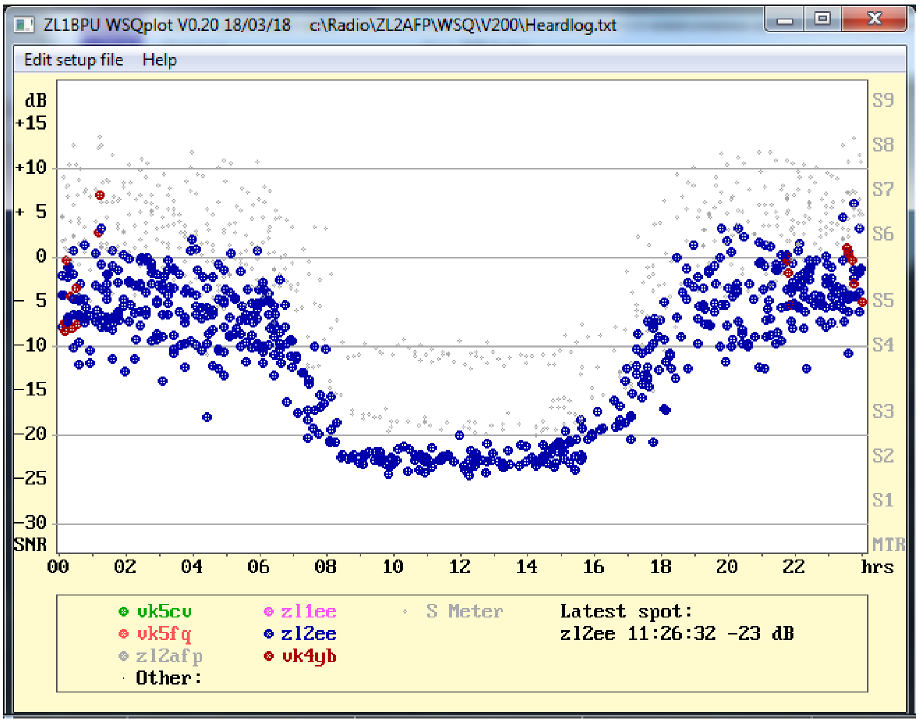


Fig. 78. The WSQCall S-Meter display

The S-Meter is dependent on receiver, sound card and software calibration, and the program now includes a gain control for this purpose.

The Heard Log now includes the S-meter readings with each station logged. Because the file format has changed, I wrote a new version of WSQplot, which has been bundled with the program. Plotting the S-meter is optional. There is a second scale to the right, which appears when S-meter plotting is enabled.



Latest spot:
zl2ee 11:26:32 -23 dB

Fig. 79. WSQPlot showing optional S-meter readings

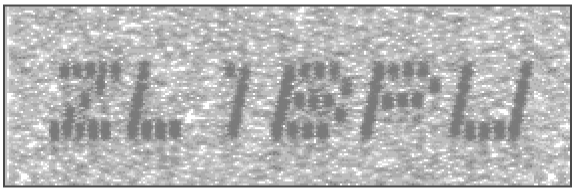
20. Where to from here?

Well, who knows where the whim will take us? You can be assured that some bright idea will come along to interest us. There may be an improvement to some existing mode, some new mode or a new application. I would like to see some of the advances in WSQCall folded back into FSQCall, although not the synchronous decoder. FSQCall needs the time flexibility of the peak hits decoder, and really doesn't need enhanced sensitivity.

Summary

So here we are at the end of 70 years of amateur radio digital mode experience and development. I've only described my own involvement – there are many other areas to be explored, and quite a few developments by other folk yet to experience and learn from.

In the 20 years since my first self-designed digital mode, MOSAIC in 1998, I've learned a tremendous amount. I don't plan to stop learning for a while yet!



Index

Alden Marinefax 9
Analogue NBTV 36
Arduino 47
ARGO 20
AMTOR 16
AVR Projects 46

Baier, Tom 51
Bangert, Ulrich 50
Baycomm 7
Bells & Whistles 64
Bold, Dr Gary 15
BMKMulty 6
Bolton, Dick 4
Büscher, Wolf 42, 51

CASTLE 47
Chip-64 29
Chirppix 42
Clicklock 39
CMSK 52
Coquelet 22
Correlogram 41
Creed 2
CSMA 55

DDS LF Exciter 32, 46
DeBene, Alberto 31
Digital NBTV 38
DominoEX 31
DominoEXChat 55
DSP56002 EVM 13
Duplo-Hell 27

Error Correction 65

FLDIGI 22
FM-Hell 26
Forrer, Johann 13
Freese, David 33, 60
FSK 2
FSQCall 59, 60
FSQ Commands 63
FSQ Development 57
FSQ Forms 67
FSQ Logging 68
FSQplot 64
FSQ Privacy 67
FSQ Telemetry 64

GPS, GPSDO 49
Gridmap 42

Hamcomm 6
Hansen, Jesper 46
Hellschreiber 16, 25
Hell DX 27
Hoff, Irving 2
Hybrid NBTV 39

Image Modes 61

Jalocha, Pawel 14, 28
JASON 31

Kantronics KPC-2 6
Kantronics UTU 7
Knowles, Bob 5

Lindecker, Patrick 33

Mancer, Eric 5
Maplin TU1000 7
Martinez, Peter 14
MEPT 19
MFSK16 21
Minchen, Ted 46
MMSSTV 15
MMTTY 15
MMVARI 15, 22
Mori, Mako 15
MOSAIC 18, 44
MSK 40
MT-Hell 19
MT63 28
Muirhead 10
MultiPSK 22

Nixie Clock 40
NOS 7

Olney, Steve 31

Piccolo 21, 58
Porcino, Nino 22, 28
PSK-Hell 26
PSKSounder 40
PSK31 14, 16

QB, QB64 48

RTTY 1, 16

SATFAX 11
Scripts 51
Sear, Lionel 18
Secondary Text 34
Siemens A2 25
Simple GPSDO 46
Soundblaster 15
Spectrogram 18
Spectrum Lab 43
STANAG 4285 38, 40
STREAM 22
Synch Decoding 73

Taylor, Joe 31
TCP/IP 7
Teletype 2
THOR 31, 35

UK-101 6

Vester, Marcus 42
VMEPT Tools 42

Wassilieff, Con 33
WEFAX 11
Western Union 5
Whale, Dr Harry 15
WSPR 31, 32
WSPRplot 69
WSQ Development 53
WSQ Varicode 53
WSQ2 56
WSQCall 70
WXSat 12

Xerox 400 5

Z38XX 50