The JT9 Coding Process (Plain Text messages only)

Andy Talbot G4JNT March 2016

This note describes my understanding of the JT9 coding process for Plain-Text messages. The first section of source coding is identical to that used for JT4, up to the end of the interleaving stage, but all has been repeated here to give a standalone description

The description in the WSJT-X user manual reads:

> *JT9 is designed for making minimally valid QSOs at LF, MF, and HF. It uses 72-bit structured messages nearly identical (at the user level) to those in JT65. Error control coding (ECC) uses a strong convolutional code with constraint length K=32, rate r=1/2, and a zero tail, leading to an encoded message length of (72+31) × 2 = 206 information-carrying bits.*
>
> *Modulation is nine-tone frequency-shift keying, 9-FSK. Eight tones are used for data, one for synchronization. Eight data tones means that three data bits are conveyed by each transmitted information symbol. Sixteen symbol intervals are devoted to synchronization, so a transmission requires a total of 206 / 3 + 16 = 85 (rounded up) channel symbols. The sync symbols are those numbered 1, 2, 5, 10, 16, 23, 33, 35, 51, 52, 55, 60, 66, 73, 83, and 85 in the transmitted sequence. Each symbol lasts for 6912 sample intervals at 12000 samples per second, or about 0.576 seconds. Tone spacing of the 9-FSK modulation is 12000/6912 = 1.736 Hz, the inverse of the symbol duration. The total occupied bandwidth is 9 × 1.736 = 15.6 Hz.*

GENJT9.EXE and its associated source code ***GENJT9.BAS*** (32 bit PowerBasic) will generate a file ***JT9SYMBS.INC*** of compressed symbols for direct import into a PIC microcontroller assembly listing.

Source Coding :

For plain text messages input data can consist only of :

(1) 10 Numbers 0 - 9
(2) 26 Upper case letters A-Z
(3) 6 Punctuation symbols [space] + - . / ?

Giving a total of 42 possible characters.

A JT9 message consists of up to13 of these characters [char 1] through to [char 13].
So, for example, a callsign and locator can be accommodated eg. ‘ G4JNT IO90IV’
(nb. There is a preceding space in this example) Shorter messages are normally padded with spaces at the end.

The message is encoded by stepping through the data and generating a value from 0 to 41 for each character in the order they are listed above, ie ‘0’ = 0, ‘9’ = 9, ‘A’ = 10, ‘Z’ = 35, ‘?’ = 41.

Three long integers are now formed as follows :

N1 = [char1]
N1 = N1 * 42 + [char 2]
N1 = N1 * 42 + [char 3]
N1 = N1 * 42 + [char 4]
N1 = N1 * 42 + [char 5]

N2 = [char6]
N2 = N2 * 42 + [char 7]
N2 = N2 * 42 + [char 8]
N2 = N2 * 42 + [char 9]
N2 = N2 * 42 + [char 10]

N3 = [char11]
N3 = N3 * 42 + [char 12]
N3 = N3 * 42 + [char 13]

N1 and N2 have a maximum value of $42^5 - 1 = 130691231$ which fits nicely into a 27 bit integer (maximum value 134217727). N3 is a 17 bit integer, having a maximum value of $42^3 = 74087$

So a total of 27 + 27 + 17 = 71 bits are needed; A single bit is added as a flag to indicate the message is plain text and gives a total of 72 bits of source data. The words are compressed and slightly rearranged into a contiguous 72 bit sequence

N1 = N1 * 2 + [bit 15] of N3 (or N3 \ 32768 MOD 2) (28 bits total)
N2 = N2 * 2 + [bit 16] of N3 (or N3 \ 65536 MOD 2) (28 bits total)
N3 = N3 MOD 32768 + 32768 (16 bits total)

Adding 32768 or setting bit 15 of N3 is the plain text flag

The contiguous 72 bit sequence is made up of $N1 * 2^{(16 + 28)} + N2 * 2^{16} + N3$
In *GENJT9.BAS the data is represented in a string format from this point on*.

31 zero bits are added at the end to make a 103 bit sequence for the next stage.

Convolutional Encoding

The data is now expanded to add FEC with a rate ½, constraint length 32, convolutional encoder.

The 103 bits (including trailing zeros) are read out MSB first:
(using the string representation, one-at-a-time from the left hand end)

The bits are clocked simultaneously into the right hand side, or least significant position, of two 32 bit shift registers [Reg 0] and [Reg 1]. Each shift register feeds an Exclusive-OR parity generator from feedback taps described respectively by the 32 bit values 0xF2D05351 and 0xE4613C47. Parity generation starts immediately the first bit appears in the registers (which must be initially cleared) and continues until the registers are flushed by the final 31st zero being clocked into them.

Each of the 103 bits shifted in generates a parity bit from each of the generators , a total of 206 bits in all. For each bit shifted in, the resulting two parity bits are taken in turn, in the order the two feedback tap positions values are given, to give a stream of 206 output bits.

The parity generation process is :

Shift the next source bit into the LSB of both [Reg 0] and [Reg 1],
moving the existing data in each one place left
Take the contents of [Reg 0]
AND with 0xF2D05351
Calculate the single bit parity (XOR) of the resulting sum.
Append to the output data stream
Take the contents of [Reg 1]
AND with 0xE4613C47
Calculate the single bit parity (XOR) of the resulting sum.
Append to the output data stream

The expansion from 72 source data bits to 206 has added sufficient redundancy in an optimised manner to give a code capable of very strong Forward Error Correction against random errors.

Interleaving

Errors over a radio link are rarely random , being more likely to occur in bursts against which this sort of convolutional coding is less effective,. So the final stage of encoding is to mix up, or interleave. the 206 data bits so as to move adjacent bits away from each other in time. The result is that close-together bits corrupted by burst interference are spread throughout the frame and therefore appear as random errors – which the FEC process can cope with.

The interleaving process is performed by taking the block of 206 starting bits labelled S[0] to S[205] and using a bit reversal of the address to reorder them, to give a pattern of destination bits referred to as D[0] to D[205].

Initialise a counter, **P** to zero
Take each 8-bit address from 0 to 255, referred to here as **I**
Bit-reverse **I** to give a value **J**.
For example, **I** = 1 gives **J** = 128, **I** = 13 **J** = 176 etc.

If the resulting bit-reversed **J** yields a value less than 206 then :
Set Destination bit D[**J**] = source bit S[**P**]
Increment **P**
Stop when **P** = 206

This completely shuffles and reorders the 206 bits on a one-to-one basis.

Up to this point, the encoding process is identical to that used for JT4 .

Convert to Symbols

Bits are taken three at a time from left to right (in a conventional representation), or in the order output from the interleaver. A value from 0 to 7 calculated from :

Bit1 * 4 + Bit2 * 3 + Bit3.
(Add a padding '0' onto the end of the 206 input bits to make a multiple of 3).

The result is an array of 69 numbers, each taking on a value from 0 to 7

Gray Coding

Input	Output
000	000
001	001
010	011
011	010
100	110
101	111
110	101
111	100

The symbols are mapped onto Gray code. Gray codes are characterised by having a change of just one bit, or interval, when moving from one symbol to the next. This minimises error propagation in the presence of noise or distortion on the signal path. The mapping is shown in the table.

This can be implemented using the function,
$N_{OUT} = N_{IN}$ ***XOR*** $(N_{IN} >> 1)$ [>> = shift right]
or by using a lookup table

Merge With Sync

16 Synchronisation symbols are now inserted amongst the 69 data symbols with a pseudo-random distribution. This is performed using the sync vector shown in the table below. This vector is 85 bits (symbols) long representing the final output stream and equates to 69 input symbols plus 16 for sync. The position of each synchronisation symbol is shown as a '1' in the pattern; the '0's reflect the position of the data symbols

Synchronisation Vector

1100100001000001000001000000000
1010000000000000011001000010000
0100000010000000000101

Stepping though the 85 symbols from left to right, if a '1' occurs a sync symbol is transmitted – this is allocated a value of 0 [zero]. If a '0' occurs in the sync vector, it is replaced by each of the successive 69 data symbols from the gray coded sequence *PLUS 1*, giving an output value for each of the 69 data symbols between 1 and 8.

So now there are 85 output symbols, each taking one of nine values from 0 to 8 .

Modulation

Since the later version of WSJT-X, a number of submodes have now appeared, as well as fast versions of the wider spaced ones. Tone spacing are based on binary multiples of 1.7361Hz (12000 / 6912) Baud rate is equal to this value for the slow modes. The fast modes are sent at speeds that are not related to this tone spacing value; The tone spacing and resulting baud rates are defined in the table below

Submode	Tone Spacing	Fast Mode Baud rate
A	1.73611	-
B	3.47222	-
C	6.94444	-
D	13.8888	-
E	27.7777	25
F	55.5555	50
G	111.111	100
H	222.222	200

Packing for export and storage

For export, the 85 symbols are packed two to a byte, MSB first, into 43 locations with appropriate formatting for PIC assembly code.

```
;   JT9 Symbols packed two per byte.  Message:  G4JNT IO90IV

    de   0x00, 0x13, 0x08, 0x84, 0x10, 0x31, 0x61, 0x40
    de   0x43, 0x47, 0x72, 0x01, 0x54, 0x16, 0x74, 0x47
    de   0x04, 0x03, 0x41, 0x83, 0x68, 0x54, 0x36, 0x86
    de   0x81, 0x00, 0x72, 0x08, 0x67, 0x70, 0x56, 0x77
    de   0x10, 0x61, 0x86, 0x63, 0x04, 0x48, 0x31, 0x52
    de   0x17, 0x07, 0x00
```