FT8: modulazione codifica e decodifica dei messaggi

Un breve percorso per capire quali sono i principi basilari
e come vengono trasmesse le informazioni nel protocollo FT8,
uno dei modi digitali che ha avuto più successo
negli ultimi anni in ambito HAM RADIO

Un po' di definizioni...

Lo scopo principale della nostra attività di radioamatori è trasmettere e ricevere informazioni via radio.
Per farlo dobbiamo generare ONDE RADIO, il che corrisponde a irradiare ENERGIA nello spazio in modo che questa possa transitare da un'antenna trasmittente a quella ricevente. Ovviamente solo una parte INFINITESIMA dell'energia irradiata in trasmissione viene trasferita all'antenna ricevente, e di qui la necessità di ottimizzare tale processo di trasferimento.

Contenuto informativo e modulazione

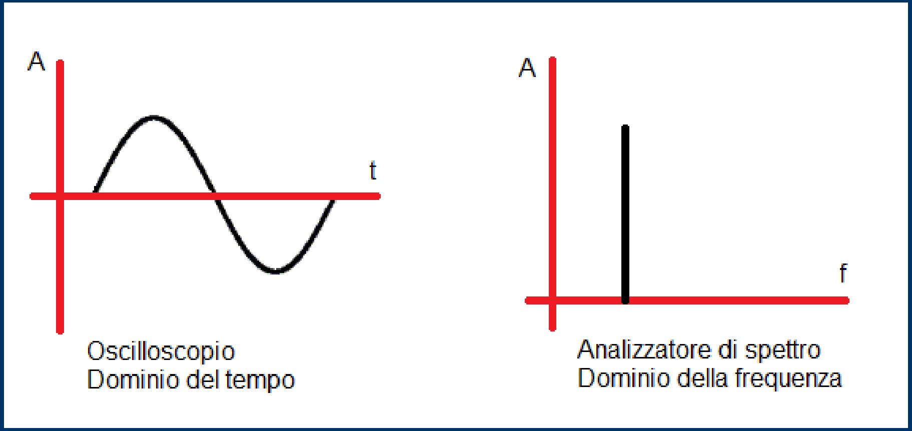
- Il nostro obiettivo è trasferire informazioni da una stazione all'altra.
- Per far questo non basta trasferire energia da una stazione all'altra, ma è necessario “caricare” tale energia di un contenuto informativo:
  - Voce
  - Dati
- Questo processo è quello che genericamente chiamiamo MODULAZIONE

Frequenza e funzioni periodiche

- Tutti sappiamo che le onde radio sono caratterizzate dalla loro FREQUENZA.
- Questo in quanto la loro propagazione è strettamente legata alla loro natura ondulatoria.
- MODULARE un'onda elettromagnetica significa caratterizzarla con un contenuto informativo che possa essere poi riconosciuto: in sintesi si tratta di “manipolare” il suo andamento.

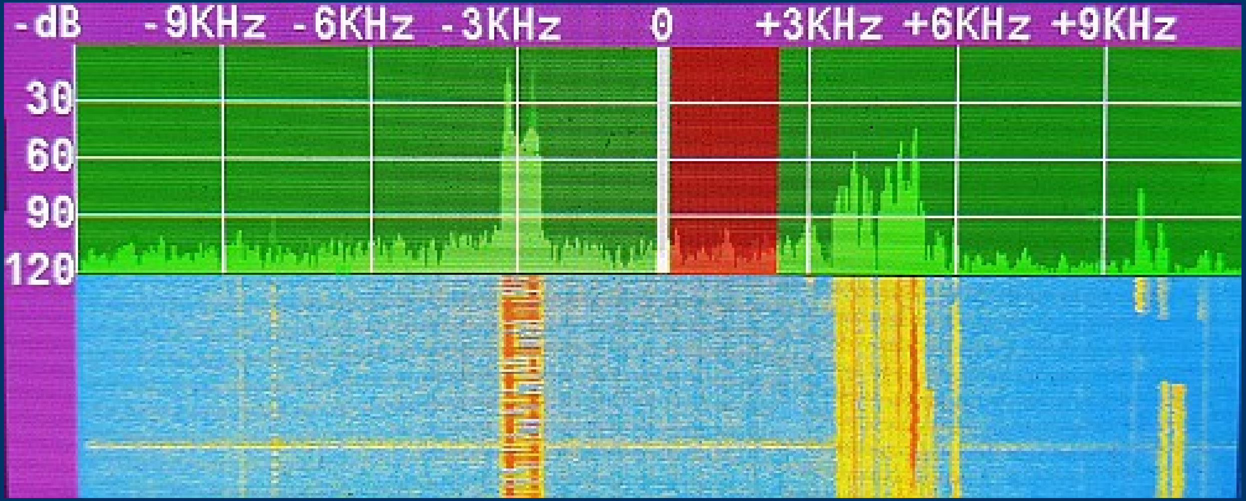
La SINUSOIDE : dominio del tempo e della frequenza

- La sinusoide è la funzione PERIODICA per eccellenza, e nel nostro caso è l'andamento che le varie grandezze elettriche che riguardano le onde radio (tensioni, correnti, campi elettrici, campi magnetici) hanno nel tempo.
- Un segnale sinusoidale, nel “dominio della frequenza”, è rappresentato da una singola barra verticale, la cui lunghezza è correlata all’ampiezza della sinusoide



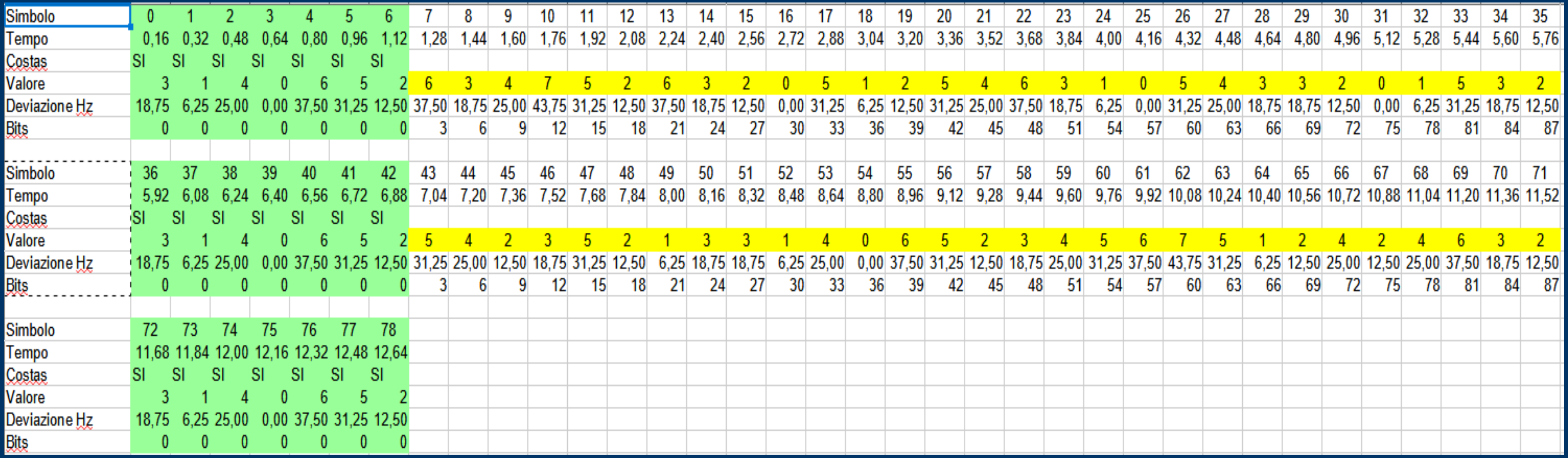
Spettro e “Waterfall”

- Alcune considerazioni:
  - Un segnale generico può essere rappresentato come una somma di sinusoidi di frequenze diverse opportunamente pesate
  - Il “peso” di ogni sinusoide rappresenta la QUANTITA' DI ENERGIA trasmessa alla specifica frequenza della sinusoide stessa
  - Il grafico dei “pesi” rappresenta quello che tutti noi conosciamo come SPETTRO



FT8 : Modulazione CPFSK a 8 toni

- Continuous Phase Frequency Shift Keys con 8 toni spaziati di 6,25 Hz (larghezza di un segnale di soli 6,25 * 8 = 50 Hz)
- 174 bits per ogni messaggio di cui:
  - 77 di effettiva informazione
  - 14 bit CRC
  - 83 bit FEC
- Ogni tono porta 3 bit di informazione (2^3 = 8): 0->000 1->001 2->010 3->011 4->100 5->101 6->110 7->111
- La durata di un tono (simbolo) è 0,16 secondi (6,25 simboli al secondo)



Simbolo	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Tempo	0,16	0,32	0,48	0,64	0,80	0,96	1,12	1,28	1,44	1,60	1,76	1,92	2,08	2,24	2,40	2,56	2,72	2,88
Costas	SI	SI	SI	SI	SI	SI	SI											
Valore	3	1	4	0	6	5	2	6	3	4	7	5	2	6	3	2	0	5
Deviazione Hz	18,75	6,25	25,00	0,00	37,50	31,25	12,50	37,50	18,75	25,00	43,75	31,25	12,50	37,50	18,75	12,50	0,00	31,25
Bits	0	0	0	0	0	0	0	3	6	9	12	15	18	21	24	27	30	33

Simbolo	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
Tempo	3,04	3,20	3,36	3,52	3,68	3,84	4,00	4,16	4,32	4,48	4,64	4,80	4,96	5,12	5,28	5,44	5,60	5,76
Costas																		
Valore	1	2	5	4	6	3	1	0	5	4	3	3	2	0	1	5	3	2
Deviazione Hz	6,25	12,50	31,25	25,00	37,50	18,75	6,25	0,00	31,25	25,00	18,75	18,75	12,50	0,00	6,25	31,25	18,75	12,50
Bits	36	39	42	45	48	51	54	57	60	63	66	69	72	75	78	81	84	87

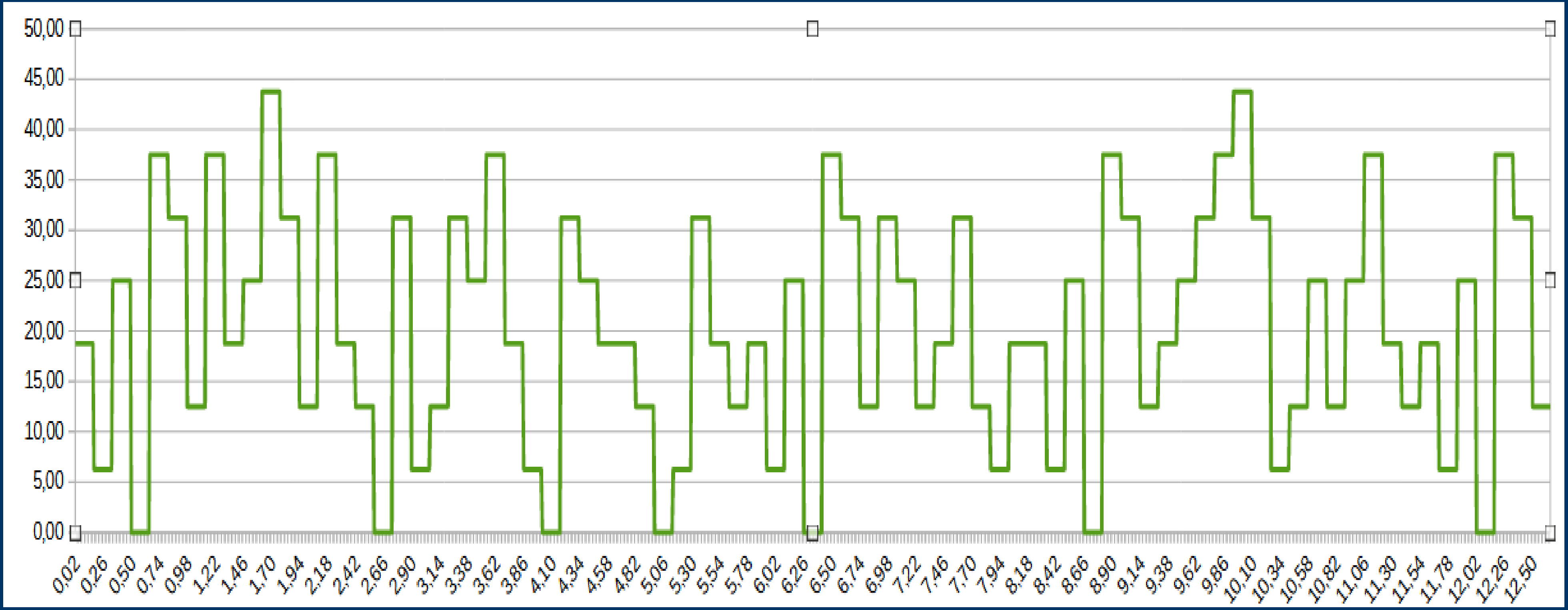
Simbolo	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53
Tempo	5,92	6,08	6,24	6,40	6,56	6,72	6,88	7,04	7,20	7,36	7,52	7,68	7,84	8,00	8,16	8,32	8,48	8,64
Costas	SI	SI	SI	SI	SI	SI	SI											
Valore	3	1	4	0	6	5	2	5	4	2	3	5	2	1	3	3	1	4
Deviazione Hz	18,75	6,25	25,00	0,00	37,50	31,25	12,50	31,25	25,00	12,50	18,75	31,25	12,50	6,25	18,75	18,75	6,25	25,00
Bits	0	0	0	0	0	0	0	3	6	9	12	15	18	21	24	27	30	33

Simbolo	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71
Tempo	8,80	8,96	9,12	9,28	9,44	9,60	9,76	9,92	10,08	10,24	10,40	10,56	10,72	10,88	11,04	11,20	11,36	11,52
Costas																		
Valore	0	6	5	2	3	4	5	6	7	5	1	2	4	2	4	6	3	2
Deviazione Hz	0,00	37,50	31,25	12,50	18,75	25,00	31,25	37,50	43,75	31,25	6,25	12,50	25,00	12,50	25,00	37,50	18,75	12,50
Bits	36	39	42	45	48	51	54	57	60	63	66	69	72	75	78	81	84	87

Simbolo	72	73	74	75	76	77	78
Tempo	11,68	11,84	12,00	12,16	12,32	12,48	12,64
Costas	SI	SI	SI	SI	SI	SI	SI
Valore	3	1	4	0	6	5	2
Deviazione Hz	18,75	6,25	25,00	0,00	37,50	31,25	12,50
Bits	0	0	0	0	0	0	0

FT8 : Modulazione CPFSK a 8 toni

- Il grafico rappresenta l’andamento della deviazione in Hz rispetto alla frequenza base (ordinata) nel tempo (ascissa)



Cosa possiamo trasmettere con 77 bit ?

Ogni “bit” è un’informazione elementare che può assumere solamente due valori:

 0 → bit spento

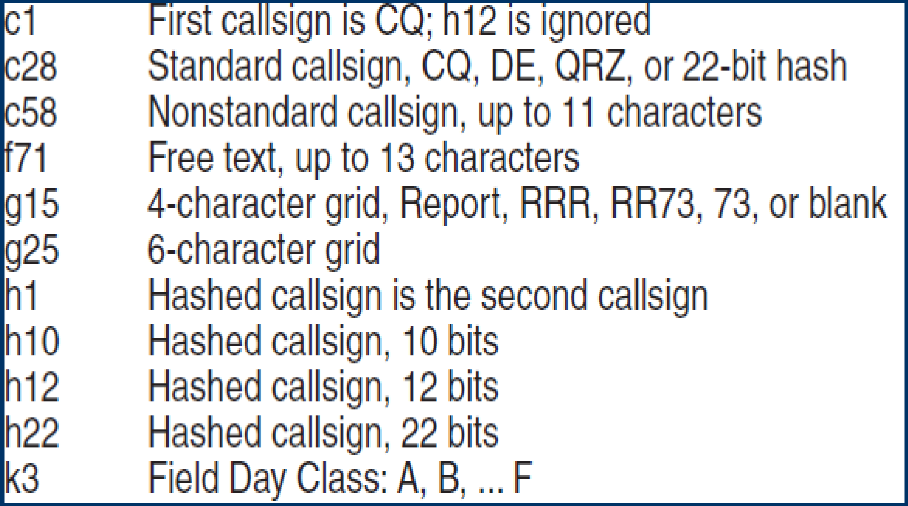
 1 → bit acceso

Se ho a disposizione più bit contemporaneamente, allora posso combinarne i valori per ottenere un maggiore dettaglio.
Ad esempio, per 2 bit posso avere 0:0=0 0:1=1 1:0=2 1:1=3 (quattro combinazioni)
In generale n bit danno origine a 2^n combinazioni, associando ad ogni combinazione un numero intero che va da 0 a 2^n – 1.
Per ottenere da questi numeri una corrispondenza ai comuni caratteri usati per la scrittura, esistono le “codifiche”.
Tra quelle più comuni citiamo:
Ascii : 8 bit (256 caratteri)
Baudot : 5 bit (32 caratteri) usati nei vecchi sistemi RTTY

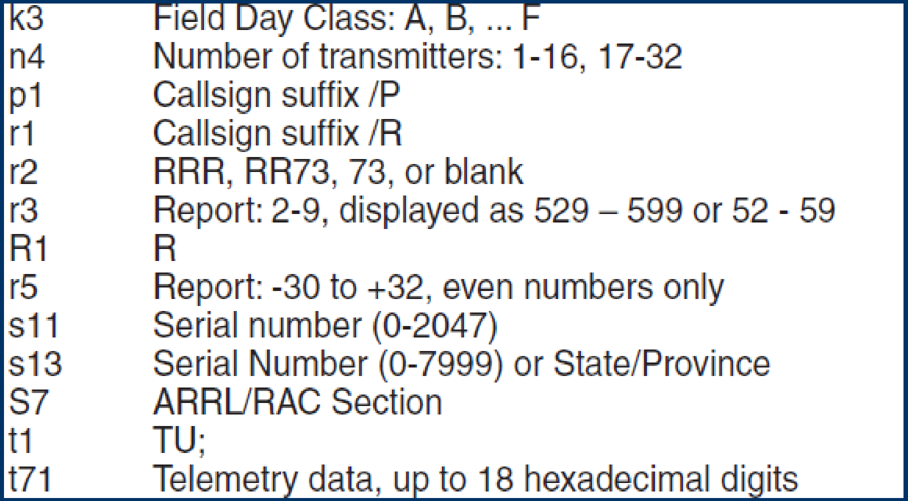
Usando una di queste codifiche ad esempio ASCII, a conti fatti, in un frammento FT8 riusciremmo a trasmettere ben poche informazioni :
77 / 8 = 9,625 cioè 9 caratteri in tutto… Troppo poco !!!

Quindi ?

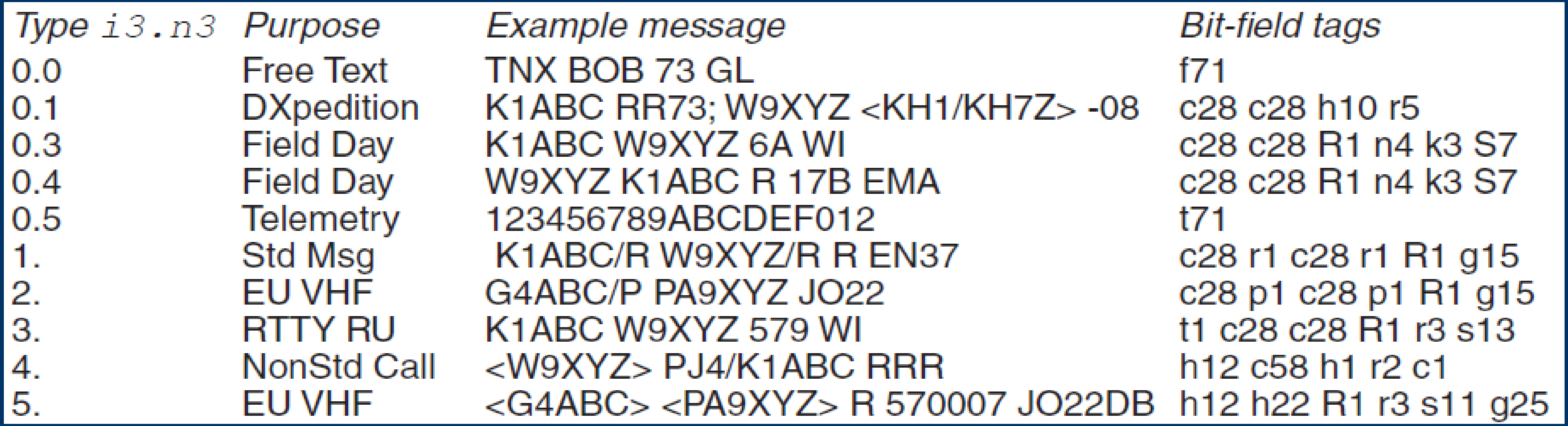
I frammenti FT8



c1	First callsign is CQ; h12 is ignored
c28	Standard callsign, CQ, DE, QRZ, or 22-bit hash
c58	Nonstandard callsign, up to 11 characters
f71	Free text, up to 13 characters
g15	4-character grid, Report, RRR, RR73, 73, or blank
g25	6-character grid
h1	Hashed callsign is the second callsign
h10	Hashed callsign, 10 bits
h12	Hashed callsign, 12 bits
h22	Hashed callsign, 22 bits
k3	Field Day Class: A, B, ... F

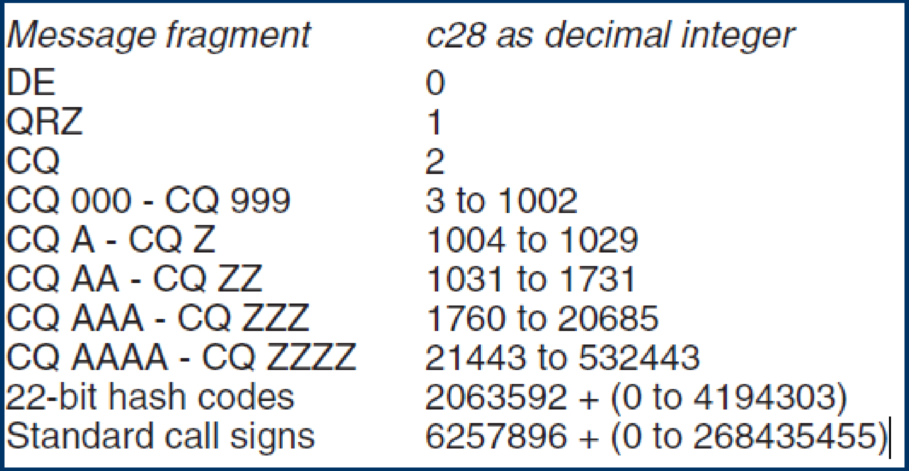


k3	Field Day Class: A, B, ... F
n4	Number of transmitters: 1-16, 17-32
p1	Callsign suffix /P
r1	Callsign suffix /R
r2	RRR, RR73, 73, or blank
r3	Report: 2-9, displayed as 529 – 599 or 52 - 59
R1	R
r5	Report: -30 to +32, even numbers only
s11	Serial number (0-2047)
s13	Serial Number (0-7999) or State/Province
S7	ARRL/RAC Section
t1	TU;
t71	Telemetry data, up to 18 hexadecimal digits



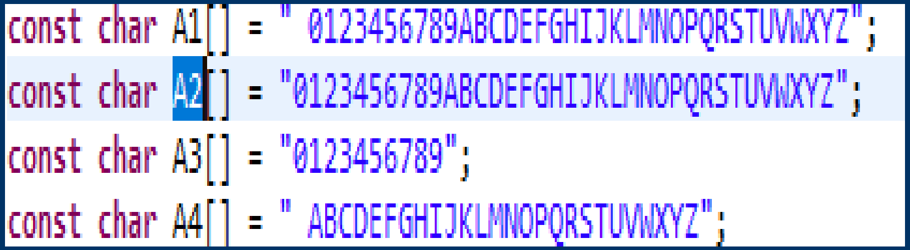
Type i3.n3	*Purpose*	*Example message*	*Bit-field tags*
0.0	Free Text	TNX BOB 73 GL	f71
0.1	DXpedition	K1ABC RR73; W9XYZ <KH1/KH7Z> -08	c28 c28 h10 r5
0.3	Field Day	K1ABC W9XYZ 6A WI	c28 c28 R1 n4 k3 S7
0.4	Field Day	W9XYZ K1ABC R 17B EMA	c28 c28 R1 n4 k3 S7
0.5	Telemetry	123456789ABCDEF012	t71
1.	Std Msg	K1ABC/R W9XYZ/R R EN37	c28 r1 c28 r1 R1 g15
2.	EU VHF	G4ABC/P PA9XYZ JO22	c28 p1 c28 p1 R1 g15
3.	RTTY RU	K1ABC W9XYZ 579 WI	t1 c28 c28 R1 r3 s13
4.	NonStd Call	<W9XYZ> PJ4/K1ABC RRR	h12 c58 h1 r2 c1
5.	EU VHF	<G4ABC> <PA9XYZ> R 570007 JO22DB	h12 h22 R1 r3 s11 g25

Il frammento c28: la codifica dei nominativi “standard”

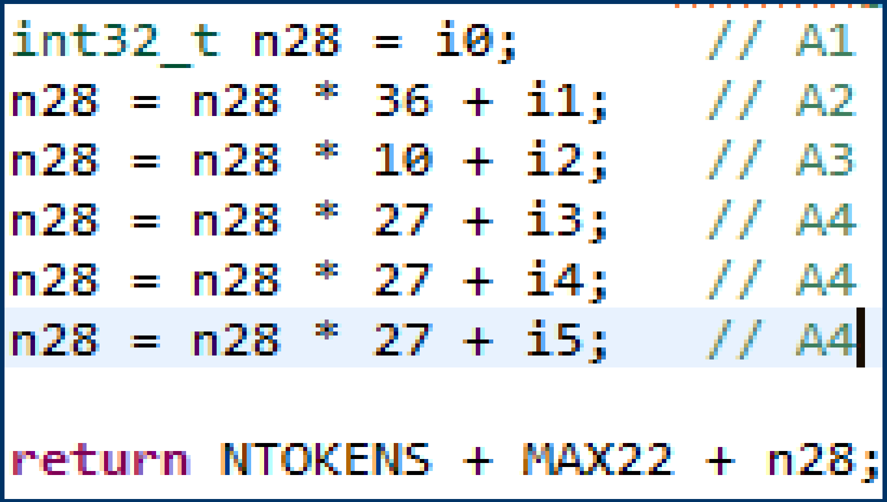


Message fragment	*c28 as decimal integer*
DE	0
QRZ	1
CQ	2
CQ 000 - CQ 999	3 to 1002
CQ A - CQ Z	1004 to 1029
CQ AA - CQ ZZ	1031 to 1731
CQ AAA - CQ ZZZ	1760 to 20685
CQ AAAA - CQ ZZZZ	21443 to 532443
22-bit hash codes	2063592 + (0 to 4194303)
Standard call signs	6257896 + (0 to 268435455)

Si noti che 2 ^ 28 = 268535456



```
const char A1[] = " 0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
const char A2[] = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
const char A3[] = "0123456789";
const char A4[] = " ABCDEFGHIJKLMNOPQRSTUVWXYZ";
```



```
int32_t n28 = i0;           // A1
n28 = n28 * 36 + i1;        // A2
n28 = n28 * 10 + i2;        // A3
n28 = n28 * 27 + i3;        // A4
n28 = n28 * 27 + i4;        // A4
n28 = n28 * 27 + i5;        // A4

return NTOKENS + MAX22 + n28;
```

- Esempio: IV3ZXF
  - I → 19 (A1)
  - V → 31 (A2)
  - 3 → 3 (A3)
  - Z → 26 (A4)
  - X → 24 (A4)
  - F → 6 (A4)
- ((((19*36+31)*10+3)*27+26)*27+24)*27+6
- 5215269 + 6257896 =**11473165**

E i nominativi "non standard" (es II3YOTA) ?

- I nominativi non standard vengono "impacchettati" in un "hash code" (...polpetta) effettuando un'operazione di somma (e altro) sui caratteri ASCII che li compongono, ottenendo un numero di 22 bit (0 - 0x3FFFFF)
- Il tutto si basa sulla bassa probabilità che due nominativi presenti nella stessa sessione abbiano lo stesso "hash"
- Sono quei nominativi che appaiono su WSJT-X con <> (<II3YOTA>)

Ma come si decodifica un messaggio ?

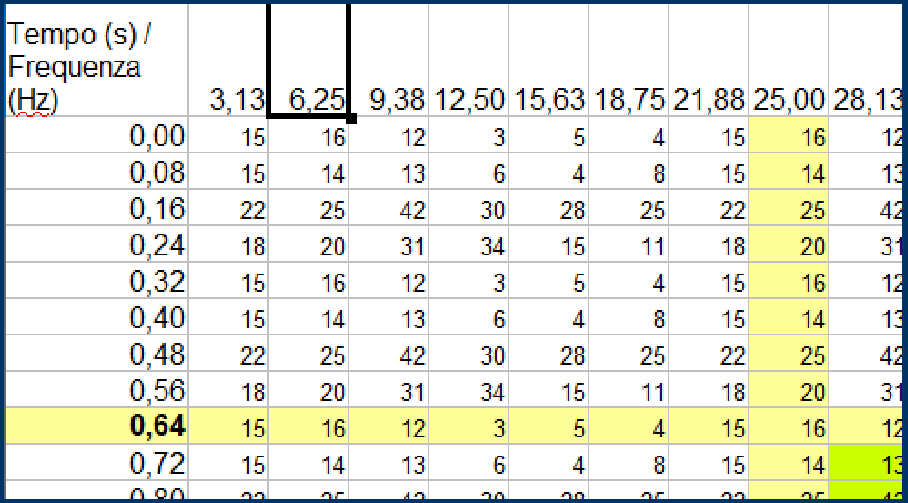
- Per decodificare un messaggio si devono identificare i 59 simboli che lo compongono e per far questo è necessario conoscere esattamente:
  - Quando inizia il messaggio
  - Qual'è la frequenza “base” su cui vengono applicati gli shift
- Da qui, per semplificare la ricerca, la decisione di definire, come noto a tutti, i 4 slot da 15 secondi nei quali collocare i messaggi FT8

FFT : Fast Fourier Transform

- Poichè il contenuto informativo sta nelle frequenze che il segnale assume nel tempo, dobbiamo eseguire un'analisi spettrale “campionando” un sufficiente numero di istanti durante i 15 secondi dello slot
- Tale campionamento si fa con un algoritmo denominato FFT che riceve in input il segnale campionato e fornisce in output le ampiezze alle varie frequenze

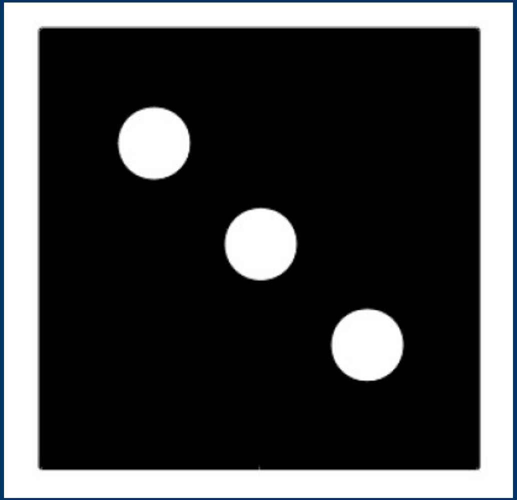
Il “TABELLONE” delle frequenze nel tempo

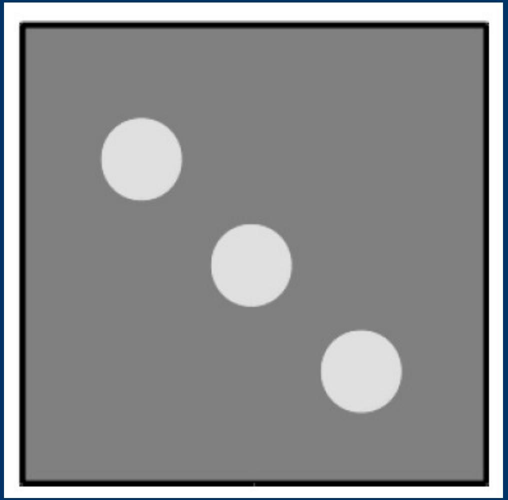
- Effettuando le FFT con opportuna periodicità, otterremo una tabella con 178 righe (una ogni 0,08 secondi ovvero 12,5 Hz, durata totale 14,24s)
- Ciascuna riga contiene 960 valori che rappresentano l'ampiezza delle componenti spettrali da 0 a 3000 Hz a step di 3,125 Hz
- Questo “TABELLONE” contiene le informazioni sui segnali di TUTTE le stazioni FT8 ricevute, per comodità già convertite in scala logaritmica (dB)

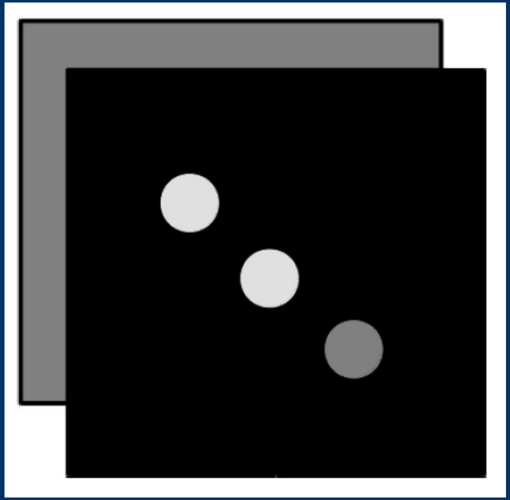


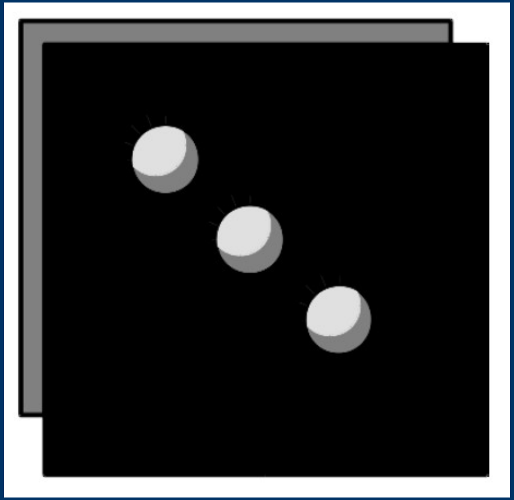
Tempo (s) / Frequenza (Hz)	3,13	6,25	9,38	12,50	15,63	18,75	21,88	25,00	28,13
0,00	15	16	12	3	5	4	15	16	12
0,08	15	14	13	6	4	8	15	14	13
0,16	22	25	42	30	28	25	22	25	42
0,24	18	20	31	34	15	11	18	20	31
0,32	15	16	12	3	5	4	15	16	12
0,40	15	14	13	6	4	8	15	14	13
0,48	22	25	42	30	28	25	22	25	42
0,56	18	20	31	34	15	11	18	20	31
0,64	15	16	12	3	5	4	15	16	12
0,72	15	14	13	6	4	8	15	14	13
0,80	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Ma come trovare i messaggi tra tutti quei numeri? COSTAS!

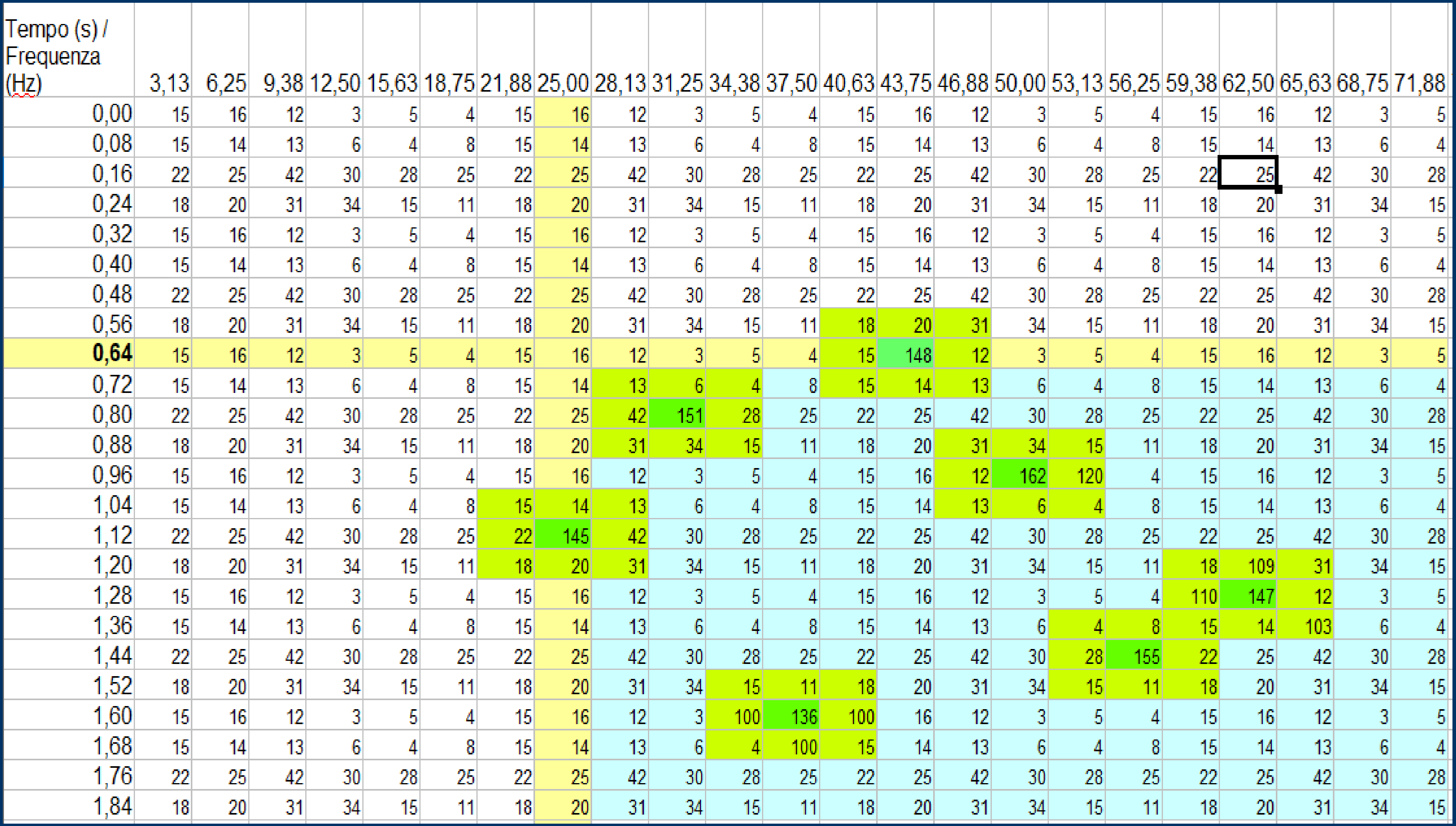








La sequenza di Costas per l'FT8: 3 1 4 0 6 5 2



Tempo (s) / Frequenza (Hz)	3,13	6,25	9,38	12,50	15,63	18,75	21,88	25,00	28,13	31,25	34,38	37,50	40,63	43,75	46,88	50,00	53,13	56,25	59,38	62,50	65,63	68,75	71,88
0,00	15	16	12	3	5	4	15	16	12	3	5	4	15	16	12	3	5	4	15	16	12	3	5
0,08	15	14	13	6	4	8	15	14	13	6	4	8	15	14	13	6	4	8	15	14	13	6	4
0,16	22	25	42	30	28	25	22	25	42	30	28	25	22	25	42	30	28	25	22	25	42	30	28
0,24	18	20	31	34	15	11	18	20	31	34	15	11	18	20	31	34	15	11	18	20	31	34	15
0,32	15	16	12	3	5	4	15	16	12	3	5	4	15	16	12	3	5	4	15	16	12	3	5
0,40	15	14	13	6	4	8	15	14	13	6	4	8	15	14	13	6	4	8	15	14	13	6	4
0,48	22	25	42	30	28	25	22	25	42	30	28	25	22	25	42	30	28	25	22	25	42	30	28
0,56	18	20	31	34	15	11	18	20	31	34	15	11	18	20	31	34	15	11	18	20	31	34	15
0,64	15	16	12	3	5	4	15	16	12	3	5	4	15	148	12	3	5	4	15	16	12	3	5
0,72	15	14	13	6	4	8	15	14	13	6	4	8	15	14	13	6	4	8	15	14	13	6	4
0,80	22	25	42	30	28	25	22	25	42	151	28	25	22	25	42	30	28	25	22	25	42	30	28
0,88	18	20	31	34	15	11	18	20	31	34	15	11	18	20	31	34	15	11	18	20	31	34	15
0,96	15	16	12	3	5	4	15	16	12	3	5	4	15	16	12	162	120	4	15	16	12	3	5
1,04	15	14	13	6	4	8	15	14	13	6	4	8	15	14	13	6	4	8	15	14	13	6	4
1,12	22	25	42	30	28	25	22	145	42	30	28	25	22	25	42	30	28	25	22	25	42	30	28
1,20	18	20	31	34	15	11	18	20	31	34	15	11	18	20	31	34	15	11	18	109	31	34	15
1,28	15	16	12	3	5	4	15	16	12	3	5	4	15	16	12	3	5	4	110	147	12	3	5
1,36	15	14	13	6	4	8	15	14	13	6	4	8	15	14	13	6	4	8	15	14	103	6	4
1,44	22	25	42	30	28	25	22	25	42	30	28	25	22	25	42	30	28	155	22	25	42	30	28
1,52	18	20	31	34	15	11	18	20	31	34	15	11	18	20	31	34	15	11	18	20	31	34	15
1,60	15	16	12	3	5	4	15	16	12	3	100	136	100	16	12	3	5	4	15	16	12	3	5
1,68	15	14	13	6	4	8	15	14	13	6	4	100	15	14	13	6	4	8	15	14	13	6	4
1,76	22	25	42	30	28	25	22	25	42	30	28	25	22	25	42	30	28	25	22	25	42	30	28
1,84	18	20	31	34	15	11	18	20	31	34	15	11	18	20	31	34	15	11	18	20	31	34	15

Il processo di decodifica di un messaggio FT8

- Grazie alla matrice di Costas (ripetuta per 3 volte all'interno del messaggio) abbiamo trovato, con una buona probabilità:
  - Lo shift di frequenza corrispondente al simbolo 0 (la frequenza più bassa)
  - Il timing esatto in cui inizia il messaggio
- Dal “tabellone” otteniamo quindi la sequenza dei 59 simboli da 3 bit ciascuno (177 bit) che contiene il messaggio e il necessario per applicare l'algoritmo di correzione dell'errore

Il controllo di validità tramite CRC

- Una volta ricavati i 177 bit, viene calcolato un CRC (Cyclic Redundance Code) e viene confrontato con il valore letto nel messaggio stesso: se il confronto è positivo, allora il messaggio è valido, altrimenti si procede a un tentativo di correzione di errore mediante la tecnica FEC, nella speranza che dopo qualche ciclo si otterrà un messaggio valido

Decodifica finale

- Dopo il controllo del CRC si ha la ragionevole certezza (probabilità tendente a 1) che il messaggio ricevuto sia valido, e si procede quindi alla sua decodifica applicando “a rovescio” le regole descritte in precedenza per la codifica del messaggio FT8

DOMANDE ?

Steve Franke, K9AN
3314 E. Anthony Dr., Urbana, IL 61802; k9an@arrl.net

Bill Somerville, G4WJS
Glendella, Wycombe Rd., Stokenchurch, High Wycombe HP14 3RP, England; g4wjs@classdesign.com

Joe Taylor, K1JT
272 Hartley Ave., Princeton, NJ 08540; k1jt@arrl.net

The FT4 and FT8 Communication Protocols

Motivation and design of the digital modes FT4 and FT8, and some details of how they are implemented in WSJT-X.

1. Introduction

FT4 and FT8 are digital protocols designed for rapid and accurate communication between amateur radio stations, particularly in weak-signal conditions. Information exchanged in a minimal two-station contact typically consists of call signs, four-character Maidenhead locators, signal reports, and acknowledgments. Special message formats support a few popular radio contests, and arbitrary text can also be conveyed, though only in small quantities. FT8 was introduced in July 2017 with version 1.8 of the software package *WSJT-X* **[1, 2]**. It quickly gained world-wide popularity, by some measures soon accounting for a large fraction of all ham radio activity on the high frequency (HF) bands **[3]**. FT4, a similar but faster protocol designed especially for radio contests, was introduced two years later in *WSJT-X* version 2.1.

The new protocols build on the legacies of JT4, JT9, JT65, and other digital modes pioneered in *WSJT-X* and its parent program *WSJT*, going back nearly 20 years **[4-11]**. These modes all use time-synchronized transmissions and structured messages with lossless compression of standard call signs, grid locators, and other basic information into a minimum number of bits. Strong forward error correction (FEC) is an integral part of each mode. FT8 and FT4 use a low density parity check (LDPC) block code designed and optimized specifically for this application.

Fundamental tenets of communication theory imply trade-offs involving message length, signaling rate, bandwidth, error-control coding, modulation type, decoding complexity, and minimum required signal-to-noise ratio (SNR). Earlier digital modes found in *WSJT-X*, including JT4, JT65, and QRA64, were optimized for extreme weak-signal performance on the VHF, UHF, and microwave bands. These modes use transmit and receive sequences of one minute, so two-way contacts generally require at least four minutes. By reducing the T/R sequences to just 15 seconds, optimizing the LDPC decoder in several important ways, and accepting a sensitivity loss (relative to JT65) around 4 dB, FT8 seems to have struck a sweet spot for DXing and general use on the HF, VHF, and lower UHF bands. FT4 accepts a further 3.2 dB loss in sensitivity but is twice as fast as FT8, and is potentially attractive for contesting at high QSO rates.

In this paper we provide full documentation of the FT4 and FT8 protocols and outline how the modes are implemented in *WSJT-X*. We present detailed performance measurements based on simulations over the additive white Gaussian noise (AWGN) channel and a range of standard International Telecommunications Union (ITU) models for HF propagation **[12]**. *WSJT-X* is an open source program licensed under version 3 of the Free Software Foundation's General Public License (GPLv3). We provide some guidelines and a few restrictions on the use of our freely available source code by others.

2. Structured Messages and Source Encoding

FT4 and FT8 transmissions always convey exactly 77 bits of user information. Mappings between human-readable messages and the underlying information bits depend on designed-for-the-purpose compression techniques known as *source encoding*. The basic aim is to give the 77-bit information payload maximum utility for conveying the information necessary for basic station-to-station contacts. To facilitate efficient message compression for a range of targeted purposes, we allocate three bits to specify one of eight possible message types. The remaining 74 bits are for user information, with source-encoding details that depend on message type. Types are tagged with an integer variable that we call `i3`, and types that require fewer than 74 information bits can use the remaining bits to define subtypes tagged as `i3.n3`.

Table 1 presents a summary of

Table 1 - Defined message types for the 77-bit payloads of FT4, FT8, and MSK144.

Type `i3.n3`	*Purpose*	*Example message*	*Bit-field tags*
0.0	Free Text	TNX BOB 73 GL	f71
0.1	DXpedition	K1ABC RR73; W9XYZ <KH1/KH7Z> -08	c28 c28 h10 r5
0.3	Field Day	K1ABC W9XYZ 6A WI	c28 c28 R1 n4 k3 S7
0.4	Field Day	W9XYZ K1ABC R 17B EMA	c28 c28 R1 n4 k3 S7
0.5	Telemetry	123456789ABCDEF012	t71
1.	Std Msg	K1ABC/R W9XYZ/R R EN37	c28 r1 c28 r1 R1 g15
2.	EU VHF	G4ABC/P PA9XYZ JO22	c28 p1 c28 p1 R1 g15
3.	RTTY RU	K1ABC W9XYZ 579 WI	t1 c28 c28 R1 r3 s13
4.	NonStd Call	<W9XYZ> PJ4/K1ABC RRR	h12 c58 h1 r2 c1
5.	EU VHF	<G4ABC> <PA9XYZ> R 570007 JO22DB	h12 h22 R1 r3 s11 g25

all currently defined message types. Each type has a defined human-readable format that is mapped into a sequence of fixed-length bit fields, each conveying specific information relevant to the type. Successive columns of **Table 1** show for each message type the number `i3` or `i3.n3`, the basic purpose, an example message, and a sequence of one or more bit-field tags. Numbers in the tag names indicate the number of bits in that field, and the total number of bits, including `n3` (if used) and `i3`, is always 77.

Table 2 identifies the type of human-readable information conveyed by each distinct bit-field tag. The tags include ones to encode call signs, Maidenhead locators, signal reports, acknowledgments, various logical flags, and permissible exchanges for a few special operating activities such as VHF contests, ARRL RTTY Roundup, and ARRL Field Day.

Standard amateur call signs can be conveyed in 28 bits, but compound calls such as PJ4/K1ABC and special-event calls like YW18FIFA may require more than twice that number. To accommodate such special calls, message type 4 allows use of one arbitrary call sign with up to 11 alphanumeric characters. The other call sign in such messages is sent as a *hash code* of just 12 bits. Such a mapping from call sign to hash code is uniquely defined, but obviously the inverse mapping cannot be unique. When the *WSJT-X* receiving software finds a message containing a hash code it displays the most recently decoded call sign mapping to that code (or perhaps "mycall", if it maps to that code), enclosed in angle brackets: for example `<PJ4/K1ABC>`. If no call sign is available for the received hash, the missing information is displayed to the user as `<...>`. Different message types may use different hash code lengths. For example, message type 0.1 always sends the transmitting station's call sign as a 10-bit hash code, and message type 1 conveys any call enclosed in angle brackets as a 22-bit hash code. Hash code "collisions" are possible, of course, but in practice we find they are rare. Further low-level details of these and other source encoding algorithms, including ranges of permitted values, can be found in Appendix A.

Table 2 - Assigned purposes for the bit fields listed in Table 1. Numbers in the tags indicate the number of bits in that field.

Tag	*Information conveyed*
c1	First callsign is CQ; h12 is ignored
c28	Standard callsign, CQ, DE, QRZ, or 22-bit hash
c58	Nonstandard callsign, up to 11 characters
f71	Free text, up to 13 characters
g15	4-character grid, Report, RRR, RR73, 73, or blank
g25	6-character grid
h1	Hashed callsign is the second callsign
h10	Hashed callsign, 10 bits
h12	Hashed callsign, 12 bits
h22	Hashed callsign, 22 bits
k3	Field Day Class: A, B, ... F
n4	Number of transmitters: 1-16, 17-32
p1	Callsign suffix /P
r1	Callsign suffix /R
r2	RRR, RR73, 73, or blank
r3	Report: 2-9, displayed as 529 – 599 or 52 - 59
R1	R
r5	Report: -30 to +32, even numbers only
s11	Serial number (0-2047)
s13	Serial Number (0-7999) or State/Province
S7	ARRL/RAC Section
t1	TU;
t71	Telemetry data, up to 18 hexadecimal digits

3. Error Detection and Error Correction

A 14-bit cyclic redundancy check (CRC) is appended to each 77-bit information packet to create a 91-bit message-plus-CRC word. The CRC is calculated on the source-encoded message, zero-extended from 77 to 82 bits. The CRC algorithm uses the polynomial 0x6757 (hexadecimal) and an initial value of zero (see **[13, 14]** for further details). Another 83 bits are appended for forward error correction, creating a 174-bit codeword.

Forward error correction is accomplished using a (174, 91) LDPC code designed specifically for FT8 and FT4. The code is defined by two matrices: a *generator matrix* used to compute the 83 parity bits appended to each 91-bit message-plus-CRC word, and a *parity matrix* that can be used to determine if a given 174-bit sequence is a valid codeword. All values in both matrices are either 0 or 1, and their related operations use modulo 2 binary arithmetic. The generator matrix has 83 rows and 91 columns. It is defined in a file `generator.dat` and included along with a number of other useful files in reference **[14]**. Nonzero values in row i of the matrix specify which of the 91 message-plus-CRC bits must be summed, modulo 2, to produce the i^{th} parity-check bit. Similarly, a file `parity.dat` **[14]** defines the sparse 83×174 parity-check matrix. The locations of the 1s in each row of this matrix specify which bits of a 174-bit codeword must sum (modulo 2) to zero. The 174-bit word is a valid codeword only if all 83 sums are zero.

4. Channel Symbols and Modulation

FT8 messages are transmitted using 8-tone continuous-phase frequency shift keying (CPFSK). Each transmitted tone or *channel symbol* conveys three bits. The sequence of 174 codeword bits is mapped onto a sequence of 174/3 = 58 channel symbols a_n, with n running from 0 to 57. The value of each symbol corresponds to a tone index in the range 0 to 7. Groups of three consecutive message bits are mapped to channel symbols using a Gray code defined by columns 1 and 2 of **Table 3**. This mapping ensures that bit triads associated with adjacent tones differ in only one bit position, thereby improving decoding performance on channels where Doppler spread is comparable to the tone separation.

FT4 is similar but uses 4-tone CPFSK, so each channel symbol conveys only two message bits. The sequence of 174 codeword bits is mapped onto a sequence of 174/2 = 87 channel symbols a_n, $n = 0, 1, 2, ..., 86$, with each symbol value an integer tone index in the range 0 – 3. Pairs of successive message bits are mapped to channel symbols according to the Gray code defined by columns 1 and 3 of **Table 3**.

Tone patterns known as *Costas arrays* are embedded in FT8 and FT4 waveforms to allow the receiving software to synchronize properly with received signals in both time and frequency. For FT8 we use the seven-tone sequence 3, 1, 4, 0, 6, 5, 2 inserted at the beginning, middle, and end of the transmitted waveform. If

Table 3 - Bi-directional Gray mapping between message bits and channel symbols.

Channel Symbol	*FT8 Bits*	*FT4 Bits*
0	000	00
1	001	01
2	011	11
3	010	10
4	110	
5	100	
6	101	
7	111	

the sync sequence is denoted S and the first and second halves of the information symbols by $M_A = \{a_0, a_1, ..., a_{28}\}$, and $M_B = \{a_{29}, a_{30}, ..., a_{57}\}$, the complete set of 79 transmitted symbols can be written as the sequence $b_n = \{S, M_A, S, M_B, S\}$.

Synchronization in FT4 uses four different Costas arrays, defined as follows:

$$S_1 = \{0,1,3,2\},$$
$$S_2 = \{1,0,2,3\},$$
$$S_3 = \{2,3,1,0\},$$
$$S_4 = \{3,2,0,1\}.$$

FT4 transmissions contain 87 information-carrying symbols. We divide these into three groups of 29:

$$M_A = \{a_0, a_1, ..., a_{28}\},$$
$$M_B = \{a_{29}, a_{30}, ..., a_{57}\},$$
$$M_C = \{a_{58}, a_{59}, ..., a_{86}\}.$$

To minimize keying transients we add a special ramp symbol R with tone index 0 that establishes slow transitions from zero amplitude at the start and to zero at the end of each waveform. A complete set of 105 channel symbols is then assembled as the sequence of values

$$b_n = \{R, S_1, M_A, S_2, M_B, S_3, M_C, S_4, R\}.$$

5. Generated Waveforms

Both protocols use continuous-phase frequency shift keying, which implies generated waveforms of the form

$$s(t) = A\cos\left(2\pi f_c t + \phi(t)\right).$$

Here A is signal amplitude, f_c is carrier frequency, t is time, and $\phi(t)$ is phase. The phase term can be written as the integral of instantaneous frequency deviation, $f_d(t)$,

$$\phi(t) = 2\pi\int_0^t f_d(\tau)\,d\tau, \quad t \geq 0.$$

This scheme guarantees that $\phi(t)$ is continuous, even across discontinuous steps in f_d. Signal amplitude A is constant, except for enforced rise and fall times at the start and end of a transmission. This constant-amplitude or *constant envelope* feature ensures that nonlinear amplification of the signal will not generate intermodulation products.

Frequency deviation $f_d(t)$ is evaluated as the weighted sum of a sequence of pulses, $p(t)$:

$$f_d(t) = h\sum_n b_n p(t - nT). \quad (1)$$

Here h is called the modulation index, weights b_n are the channel symbol values, $p(t)$ is the frequency deviation pulse shape, and T the signaling interval. (Note that T is the inverse of the keying rate, usually measured in bauds.) The frequency deviation pulse is normalized to have unit area,

$$\int_{-\infty}^{\infty} p(t)\,dt = 1, \quad (2)$$

so the pulse weighted by symbol b_n causes the carrier phase to advance by $2\pi h b_n$ radians over the duration of the pulse. For *WSJT-X* modes other than FT4 and FT8, $p(t)$ is a rectangular pulse with duration T and peak amplitude $1/T$. For these modes the instantaneous frequency deviation of the n^{th} pulse is hb_n/T.

The pulse amplitudes, b_n, are integers in the range 0 – 7 for FT8 and 0 – 3 for FT4. With this convention the carrier frequency f_c is the lowest tone frequency, i.e., the frequency transmitted during an interval with $b_n = 0$. The occupied bandwidth and spectral shape of the CPFSK signal depend on the modulation index, h, the signaling interval T, and the shape of the frequency deviation pulse $p(t)$. These parameters are summarized in **Table 4**, where for comparison we include, in addition to FT4 and FT8, several other modes supported in *WSJT-X*. The modes JT4, JT9, JT65, and MSK144 use rectangular frequency-deviation pulses, so their frequency deviations f_d have discontinuous jumps when b_n changes from one signaling interval to the next. Such jumps create relatively large spectral sidelobes in the CPFSK signal. In FT4 and FT8 the sidelobes are minimized by smoothing the frequency deviation function $f_d(t)$ with a Gaussian filter. Waveforms generated in this way are called Gaussian Frequency Shift Keyed (GFSK) signals.

Smoothing the frequency deviation waveform directly would require a numerical convolution every time a new message is generated. It is computationally more efficient to apply the Gaussian smoothing filter to an isolated rectangular pulse, which needs to be done only once. The Gaussian-smoothed pulses can then be superposed according to Equation (1) to obtain the GFSK frequency deviation waveform. Such waveforms are equivalent to those obtained by first generating the FSK waveform with rectangular pulses and then convolving it with a Gaussian smoothing function.

The GFSK Gaussian-smoothed pulse satisfying the normalization condition (2) can be written in terms of the error function, erf(x), as

$$p(t) = \frac{1}{2T}\left[\operatorname{erf}\left(kBT\left(\frac{t}{T}+0.5\right)\right) - \operatorname{erf}\left(kBT\left(\frac{t}{T}-0.5\right)\right)\right]. \quad (3)$$

Here the constant $k = \pi\sqrt{2/\ln 2} = 5.336...$, and B is the smoothing filter's -3 dB bandwidth. The error function erf(x) is defined as

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}\,dt.$$

A Gaussian smoothing filter has a low-pass frequency response with Gaussian shape. For FT8 we set the -3 dB bandwidth to $B = 2T^{-1}$, or $BT = 2$. FT4 uses a more heavily smoothed pulse with

Table 4 - Parameters of the CPFSK signals used for six modes in *WSJT-X*. Pulse amplitudes assume integer values over the specified ranges.

Mode	*T(s)*	*Modulation index, h*	*Pulse shape, p(t)*	*Pulse amplitude, b_n*
FT4	0.048	1	Gaussian-smoothed, BT=1	0 – 3
FT8	0.160	1	Gaussian-smoothed, BT=2	0 – 7
JT4	0.2286	1	Rectangular	0 – 3
JT9	0.5760	1	Rectangular	0 – 8
JT65	0.3715	1	Rectangular	0, 2 – 65
MSK144	0.0005	0.5	Rectangular	0 – 1

$BT = 1$. In the limit of very large filter bandwidth, $BT \gg 1$, the pulse shape $p(t)$ defined by Equation (3) becomes rectangular and equivalent to the pulse used to generate standard FSK as in the older *WSJT* modes. The frequency deviation pulses actually used in *WSJT-X* are plotted for comparison in **Figure 1**.

Duration of the smoothed FT4 and FT8 pulses is greater than the signaling interval T, so successive pulses overlap. This intersymbol interference (ISI) is introduced deliberately for the purpose of shaping the emitted spectrum to reduce sidelobe levels. For the values of BT used in FT4 and FT8, significant ISI is present only between immediately adjacent pulses. Thus, when summing pulses to produce $f_d(t)$ using Equation (1), at time t it is only necessary to include contributions from the pulse whose center is closest to t and those immediately before and after it.

Figure 2 shows a portion of an FT4 frequency deviation waveform $f_d(t)$. It's easy to see that corners of the waveform at symbol boundaries have been rounded. As an example to illustrate the reduction in spectral sidelobes effected by using GFSK, **Figure 3** shows the spectrum of an FT4 signal and one generated using standard FSK with no smoothing. The difference in sidelobe levels is striking. The compact spectra of FT4 and FT8 signals make it possible for dozens of them to occupy a spectral slice of a few kHz, with little or no inter-signal interference.

FT4 and FT8 waveforms have constant amplitude except at the very beginning and end of a transmission. FT8 signals are ramped up gradually over a transition interval $T/8$, or 20 ms, at the beginning of the first sync symbol. The ramp function is a raised cosine,

$$A(t) = 0.5\left[1 - \cos(8\pi t / T)\right], \qquad 0 \le t \le T/8.$$

Figure 4 illustrates the FT8 ramp function and the leading part of an FT8 waveform. The same taper is used in reverse to gradually ramp down the signal at the end of the transmission. FT4 waveforms are similarly soft-keyed, this time using raised-cosine tapers applied over the full 48 ms duration of the special ramping symbols R, described earlier.

6. Symbol Detection and Decoding

Sections 2 through 5, Appendix A, and resources in reference **[14]** define the source encoding, error-correction coding, and modulation scheme associated with the FT4 and FT8 protocols. Any proper implementation of these protocols should strictly adhere to these definitions. In this section we turn to a discussion of some implementation-specific methods used in *WSJT-X* for detecting

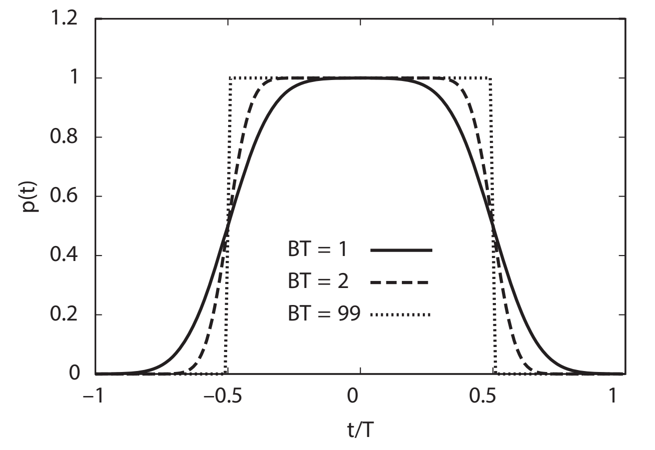


Figure 1 — Gaussian-smoothed frequency deviation pulses. The pulses labeled BT=1 and BT=2 are used to generate the FT4 and FT8 frequency-deviation waveforms, respectively. The case BT=99 is essentially the same as the unfiltered rectangular pulse that is used to generate standard, unsmoothed FSK in JT65, JT9, and other modes in *WSJT-X*.

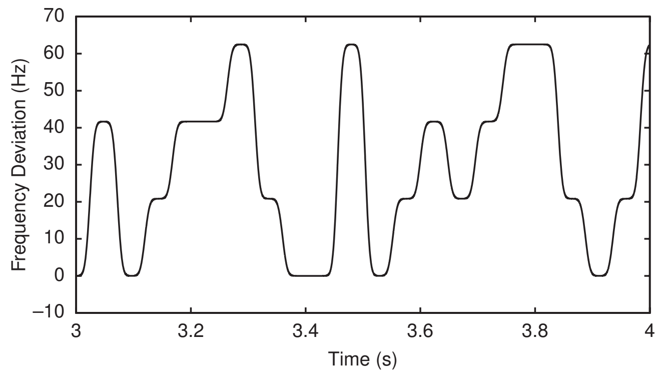


Figure 2 — A segment of an FT4 frequency deviation waveform generated using the Gaussian-smoothed frequency deviation pulse with BT=1.0.

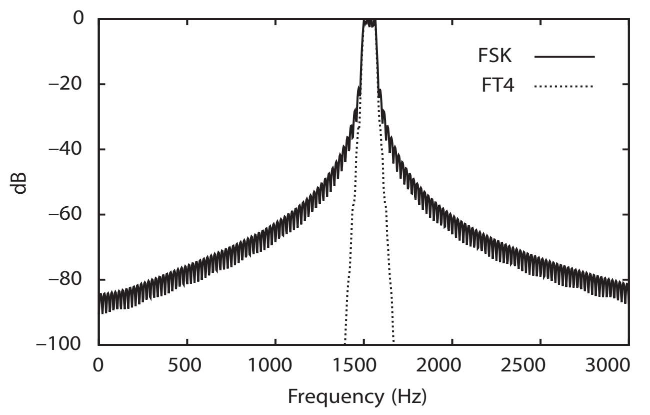


Figure 3 — Average spectrum of an FT4 signal (GFSK, BT=1.0: dotted line) and the spectrum of an otherwise equivalent standard FSK waveform (solid line).

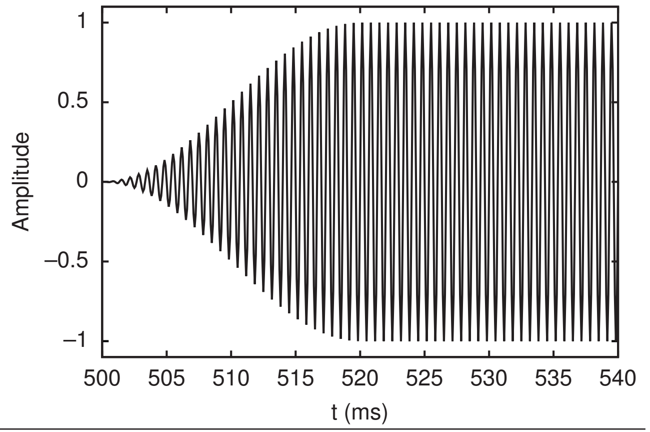


Figure 4 — The beginning of an FT8 waveform showing gradual rise of the signal envelope over the first 20 ms. FT4 uses a similar raised-cosine ramp expanded to 48 ms duration. Time-reversed ramp-down functions are used at the ends of transmissions.

and decoding FT4 and FT8 signals. Other approaches are certainly possible, and might be used by other software developers.

WSJT-X uses advanced detection and decoding techniques to decode the weakest possible signals. *Noncoherent block detection* over sequences of two or more channel symbols improves sensitivity over single-symbol detection when the received signal maintains phase coherence over multiple symbols **[15]**. Single-symbol detection provides robust detection on rapidly fading channels, while longer block lengths provide better sensitivity in slow fading conditions. We use block lengths of $N = 1, 2$, and 3 symbols for FT8, and $N = 1, 2$, and 4 for FT4.

Block detection is carried out by correlating received waveform segments spanning N symbols with locally generated waveforms corresponding to each of M^N possible symbol sequences. Here M is the number of different modulation waveforms (tones) used for each mode, $M = 8$ for FT8 and $M = 4$ for FT4. For this technique to provide sensitivity gains the received signal must be phase-stable over the block length. Such detection is said to be *noncoherent*, however, because phase continuity between sequences is not assumed. Only the magnitude of the complex-valued correlation is used, and detection is independent of phase differences between the received signal and locally generated waveforms. Output from the block detector is a set of M^N positive real correlation values. These are used to derive soft decisions for each of the $N \log_2 M$ bits conveyed by the N- symbol sequence.

Conversion of the waveform correlations to soft decisions is accomplished using a *soft demapper*. The soft metric L_j for x_j, the j^{th} bit associated with a particular symbol sequence, can be written as

$$L_j = K\left(\max_{i:x_j=1}|C_i| - \max_{i:x_j=0}|C_i|\right),$$

where $\max_{i:x_j=1}|C_i|$ is the largest correlation magnitude from the set of $M^N/2$ correlations between the received waveform and ideal waveforms associated with bit sequences having $x_j = 1$. Likewise, the second term is the largest magnitude from the other half of the set of correlations, associated with $x_j = 0$. The normalization constant, K, is adjusted empirically to optimize performance over a range of SNRs and channel conditions. For example, consider block detection of sequences of $N = 3$ successive FT8 symbols. Since each symbol could have any one of $M = 8$ values, there are $8^3 = 512$ possible three-symbol waveforms to be correlated with each three-symbol block of the received waveform. The resulting 512 correlation values determine soft decisions for the 9 bits conveyed by the sequence. In practice, *WSJT-X* calculates the 8 correlations required for single-symbol ($N = 1$) detection first, using a fast Fourier transform. The single-symbol complex results are then combined to produce the 64 correlations required for $N = 2$ and the 512 correlations required for $N = 3$ block detection, thereby avoiding needless redundant calculations.

A set of 174 soft decisions is obtained for each block size. These are submitted to the decoder, starting with the set for $N = 1$. If the decoder returns a codeword whose 77-bit message produces a 14-bit CRC that matches the decoded CRC, the algorithm terminates and the decoded message is unpacked and displayed to the user.

Our decoder uses a hybrid approach that combines the fast, iterative, *belief propagation* (BP) algorithm **[16]** with the more sensitive but computationally expensive *ordered statistics decoding* (OSD) algorithm **[16]**. In practice we find that most received signals are decoded with just a few iterations of the BP algorithm. When BP fails to find a valid codeword after a reasonable number of iterations, the soft decisions are submitted to the OSD algorithm. The OSD algorithm can provide several dB of additional sensitivity under certain conditions. Our implementation of the OSD algorithm incorporates the preprocessing rules from reference **[17]**. We found that these shortcuts significantly improve the computational efficiency of the OSD approach.

Our decoders can optionally make use of so-called *a priori* (AP) information as it accumulates during a QSO. For example, when you answer a CQ you already know your own call sign, that of your potential QSO partner, and the type of message you expect to receive if your reply is successful. Your decoding software therefore "knows" what might be expected for at least 62 message bits (29 for each of two call signs and /R flags, 1 for the R indicator, 3 for message type) in the next received message. With these bits hypothesized as a conjecture, the decoder's task can be reduced to determining the remaining 15 bits of the message and ensuring that the resulting solution is reliable. The use of AP information can increase sensitivity by several dB at the cost of a somewhat higher rate of false decodes.

Table 5 shows measured decoding thresholds for FT4 and FT8 derived from simulations, providing insight into the sensitivity gains offered by block detection and OSD. Here and elsewhere in this paper, quoted sensitivity thresholds are the signal-to-noise ratios in 2500 Hz bandwidth at which decoding probability is 0.5. Numbers in the first table row for each mode represent a baseline case with single-symbol $N = 1$ detection and BP decoding. The next two rows add block detection and block detection with hybrid (BP and OSD) decoding. Column 2 gives results for the non-fading additive white Gaussian noise (AWGN) channel, while columns 3 and 4 are for channels with frequency spreads of 1 Hz and 10 Hz, respectively. See Section 8 for more information about the channel models. One can see that on the AWGN channel block detection improves sensitivity by 1.6 dB and 0.7 dB for FT4 and FT8, respectively. Addition of hybrid decoding provides another 0.6 dB and 0.5 dB. Overall, block detection and the hybrid decoder offer 2.2 and 1.2 dB of sensitivity improvement over the baseline case for FT4 and FT8, respectively. On the mid-latitude disturbed channel, block detection improves sensitivity by 1.3 dB and 0.5 dB, and overall improvements are 2.5

Table 5 - Decoding thresholds for three different channels and three decoding schemes.

Decoding Algorithm	*AWGN (dB)*	*Mid-latitude Disturbed (dB)*	*High-Latitude Moderate (dB)*
FT4:			
N=1; BP	–15.3	–12.7	–10.4
N=1,2,4; BP	–16.9	–14.0	–10.5
N=1,2,4; BP+OSD	–17.5	–15.2	–12.2
FT8:			
N=1; BP	–19.6	–16.5	-
N=1,2,3; BP	–20.3	–17.0	-
N=1,2,3; BP+OSD	–20.8	–18.6	–8.6

and 2.1 dB. Doppler spread on the high-latitude disturbed channel exceeds the FT8 tone spacing, and FT8 fails to achieve a 50% decode probability at any SNR using only the baseline decoding scheme. With its larger tone spacing FT4 outperforms FT8 on this channel. Block detection does not offer significant improvement in this case — only 0.1 dB for FT4 — because coherence time on this rapidly fading channel is comparable to or smaller than the durations of the multi-symbol blocks.

After a signal is decoded we have the information necessary to regenerate the waveform that was actually transmitted. We use the regenerated signal as a reference to derive the time-varying, complex gain function that describes the effect of the propagation channel. Denote the real received audio signal by *s*(*t*) and define a complex reference signal by

$$r(t) = A(t)e^{j(2\pi f_0 t + \phi(t))} ,$$

where f_0 is the estimated audio frequency of the decoded signal and *A*(*t*) and $\phi(t)$ are the amplitude and phase of the transmitted waveform. The time-varying complex channel gain function, *g*(*t*), is obtained via:

$$g(t) = LPF\left[s(t)r^*(t)\right],$$

where *LPF*[ ] represents a low-pass filter and * represents the complex conjugate. The low-pass filter cutoff is optimized for best performance with real signals received under a range of different propagation conditions. The estimated channel gain function is applied to the ideal reference signal to reconstruct a nearly noiseless version of the received signal's waveform, including channel-induced amplitude fading and phase variation. The reconstructed signal is then subtracted from the received data, i.e.

$$s'(t) = s(t) - 2\Re\left[g(t)r(t)\right]$$

where $\Re[\,]$ takes the real part of its argument, and *s'*(*t*) is the audio waveform after subtracting the decoded signal. This subtraction process can uncover weaker signals that occupy the same frequency slot as the subtracted strong signal. The weaker signals can often be decoded on a second decoding pass, after all signals decoded in the first pass have been subtracted.

WSJT-X analyzes an audio waveform that may contain many signals in the received passband. A decoding pass starts by identifying all likely signals, or *candidates*, using spectral analysis. Then, for each candidate in succession, our procedure is to (1) *synchronize*, estimating the frequency offset from the upper-sideband dial frequency and the time offset with respect to the computer's system clock; (2) determine soft decisions for each of three block lengths and attempt to decode each one using either BP alone or BP+OSD; (3) if the decode is successful, subtract the signal. If at least one signal is decoded and subtracted in the first decoding pass, the remaining audio waveform is re-analyzed. New candidates are identified and steps 1 through 3 are carried out for each one. If at least one new signal is decoded and subtracted in the second pass, a third pass will sometimes yield decodes missed in the first two passes. Multi-pass decoding has proven very effective: the approach is often able to decode two or three signals at the same or nearly the same frequency.

Acronyms and abbreviations used in text.

AP	*a priori,* as in AP decoding
AWGN	additive white Gaussian noise
BP	belief propagation
CPFSK	continuous-phase frequency-shift keying
CRC	cyclic redundancy check
FEC	forward error correction
GFSK	Gaussian frequency-shift keying
GPLv3	General Public License, version 3
ISI	inter-symbol interference
ITU	International Telecommunication Union
LDPC	low-density parity check code
MinGW	Minimalist GNU for Windows
OSD	ordered statistics decoding
SNR	signal-to-noise ratio
UI	user interface

7. Message Sequencing

A basic QSO between K1JT and K9AN might consist of the following messages:

```
Tx6:  CQ K1JT FN20
                          Tx1:  K1JT K9AN EN50
Tx2:  K9AN K1JT -10
                          Tx3:  K1JT K9AN R-12
Tx4:  K9AN K1JT RRR
                          Tx5:  K1JT K9AN 73
```

Here the Txn: labels are those used to identify message entry fields on the *WSJT-X* user interface. In this model QSO even-numbered messages are sent by K1JT, odd-numbered messages (those displaced to the right) by K9AN. Note that all messages bear the transmitting station's call sign, and all but the CQ are explicitly directed to a particular QSO partner. Most QSOs follow the sequence shown, or a closely related one depending on context, propagation, and interference conditions. In normal usage, failure to decode a response from a QSO partner implies repetition of the previous transmission.

Traditionally, operators have clicked buttons on the user interface (UI) of *WSJT-X* **[1, 2, 19]** and its predecessor *WSJT* **[4-11]** to trigger transmission of the next message in a sequence similar to that shown above. Protocols with one-minute T/R sequences leave about 10 seconds between decoding and start of the next transmission, so there is plenty of time to decide upon and activate one's next transmission. However, the shorter sequences of FT8 and FT4 require much quicker responses, effectively requiring moderate amounts of automation built into the software.

If a UI checkbox labeled **Auto Seq** has been ticked, the software parses decoded messages to decide whether a valid response for the standard QSO sequence has been received, directed to "mycall". If so, the next message in the sequence is queued and transmitted; if not, the previous transmission is repeated. This sequence of events proceeds until a QSO is completed or abandoned by user action. With a few optional exceptions ("Fox" operation in FT8 DXpedition mode, and contest operation), user action is also required to verify QSO details and submit them for logging. In any event, by design *WSJT-X* requires that every QSO must be initiated by a human operator. We do not like the idea of fully robotic operation with modes like FT4 and FT8, and the *WSJT-X* software prevents it.

These moderate levels of automation involve considerably more logic in the software, essentially duplicating the thought processes of an attentive user progressing through a valid QSO. In typical operation on a busy HF or VHF band, dozens of valid FT4 or FT8 messages may be decoded in a single receiving sequence. The receiving software must analyze all of these, determine whether any is a valid reply in the expected sequence for a QSO already underway, and take appropriate action. **Figure 5** provides a high-level summary of the six major states of the *WSJT-X* auto-sequencing behavior. On program startup, and when **Halt Tx** is clicked, the logical *state machine* is put into the Calling state. Subsequent stages of a standard

QSO pass through the Replying, Report, Roger_Report, Rogers, and Signoff states while proceeding through the sequence of transmitted and received messages illustrated at the start of this section.

Each major state has its own internal logic that determines what action will be taken when specified types of messages are received. An idealized and abbreviated representation of this logic for the Calling state is presented in **Figure 6**. Here and in **Figure 5** we have used Unified Modeling Language notation **[18]**, which should be mostly self-explanatory. For example, the text

[message == Tx1] / Tx message = Tx2

attached to a state transition line has the form "trigger [guard condition] / action," so this is a transition that happens automatically without any triggering event, but only if the just parsed message has the standard Tx1 format. In this case the transition has a side effect of setting the next message queued for transmission as the generated Tx2 message.

The user interface provides options for configuring various details of the auto-sequencing behavior; please consult the *WSJT-X 2.1 User Guide* **[19]** for further details.

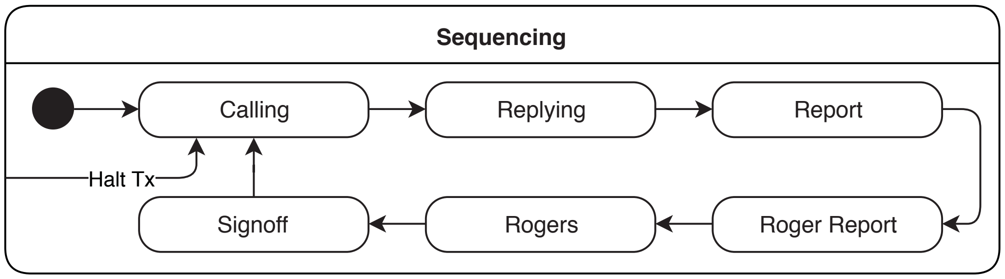


Figure 5 — Top-level state diagram for standard message sequencing in WSJT-X.

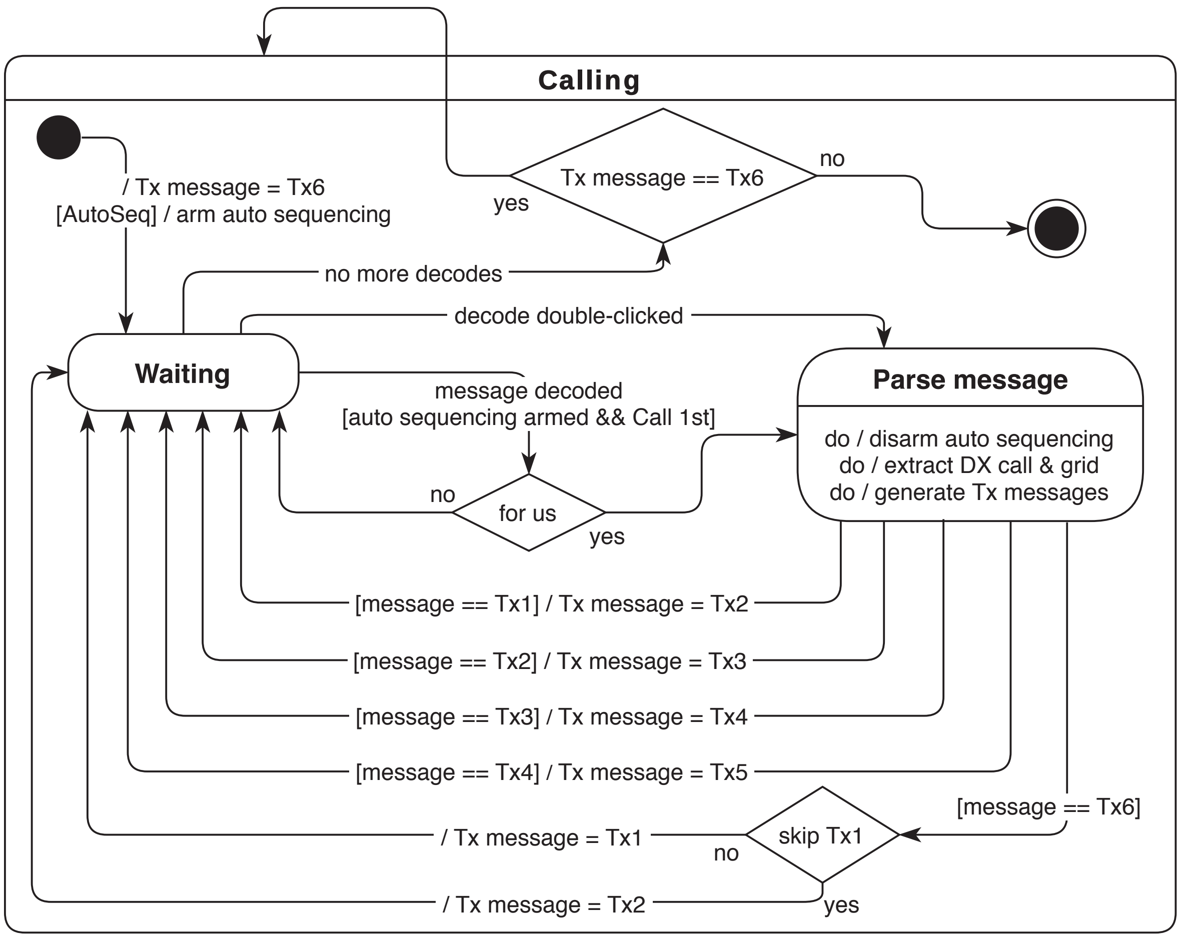

Calling
Tx message == Tx6
no
yes
/ Tx message = Tx6
[AutoSeq] / arm auto sequencing
no more decodes
decode double-clicked
Waiting
Parse message
message decoded
[auto sequencing armed && Call 1st]
do / disarm auto sequencing
do / extract DX call & grid
do / generate Tx messages
no
for us
yes
[message == Tx1] / Tx message = Tx2
[message == Tx2] / Tx message = Tx3
[message == Tx3] / Tx message = Tx4
[message == Tx4] / Tx message = Tx5
[message == Tx6]
/ Tx message = Tx1
skip Tx1
no
/ Tx message = Tx2
yes

Figure 6 — Example detailed state chart for the high-level "Calling" state shown in Figure 5.

8. Performance Measurements

While designing FT4 and FT8 we used channel simulations based on the approach recommended by the ITU **[12]** to assess the performance that could be achieved under real-world conditions. The channel model assumes two independently fading paths with equal mean attenuation and equal frequency spreads. A channel is defined by its frequency spread and differential path delay. Reference **[12]** specifies frequency spread and path delay parameters for quiet, moderately disturbed, and highly disturbed conditions in low-latitude, mid-latitude, and high-latitude terrestrial regions. The ITU channels referenced in this paper have differential path delays that are small compared to FT4 and FT8 symbol durations, so this parameter has little effect on performance. The size of a channel's frequency spread relative to the tone spacing of the mode is the most important factor determining the extent of performance degradation.

Figure 7 compares FT4 and FT8 performance obtained using *WSJT-X* version 2.1 to decode simulated FT4 and FT8 waveforms under non-fading additive white noise (AWGN) conditions, and on *mid-latitude disturbed* and *high-latitude moderately disturbed* channels. **Table 6** lists the channel parameters and the decoding threshold for these channels along with the *mid-latitude quiet* and *mid-latitude moderate* channels. FT8 maintains about 3 dB sensitivity advantage over FT4 on AWGN and mid-latitude channels where frequency spread is smaller than the tone spacing for both modes. Frequency spread on the high-latitude channel is larger than the FT8 tone spacing and smaller than FT4's tone spacing, which explains why FT4 performs best in that case.

Table 6 includes decoding thresholds obtained when the maximum possible amount of AP information is available. Maximum AP information provides about 2 dB of additional sensitivity on less disturbed channels and increasing improvement as frequency spread increases. On the most disturbed channel, large improvements of about 5 dB and 10 dB are obtained for FT4 and FT8, respectively. The trend is similar for scenarios where less AP information is available.

Figure 8 provides a detailed look at how using AP information can improve sensitivity as a QSO progresses. The *mid-latitude disturbed* channel was used for this set of simulations. The curve labeled "no AP" is the same as the curve labeled "FT8 Mid disturbed" **Figure 7**. The curve labeled "caller" shows sensitivity to responses to a CQ, or any other message directed to "mycall". The curve labeled "report" indicates sensitivity to messages directed to "mycall", originating from the known call sign of a QSO partner. This curve represents the sensitivity to a report or roger-report message during a QSO. Finally, the curve labeled "RRR/RR73/73" represents sensitivity to an RRR, RR73, or 73 message from a QSO partner, at the end of a QSO.

Figure 8 shows a modest sensitivity improvement to a caller when only "mycall" is used as AP information. The advantages of such AP information become more significant when the caller fails to start transmitting at the nominal start time, 0.5 s after the

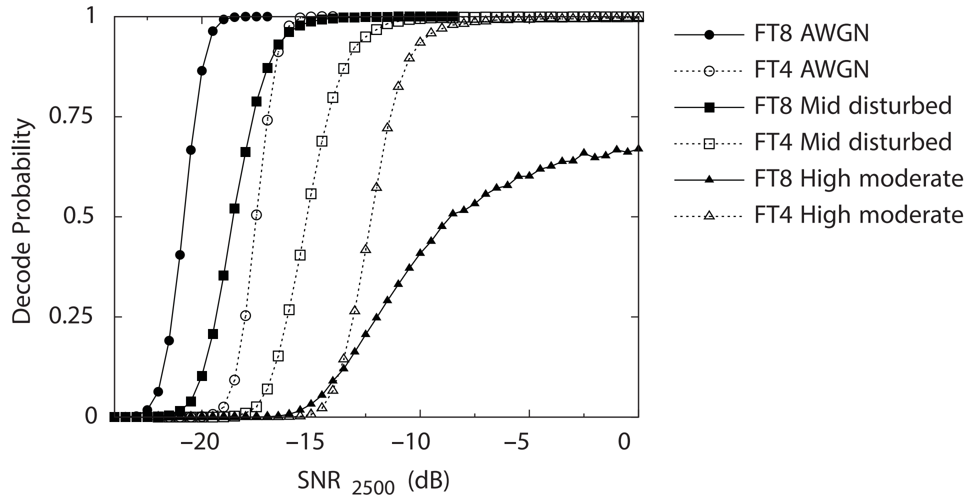


Figure 7 — Measured decoding probability as a function of SNR for FT8 and FT4, based on simulations for three propagation channels: additive white Gaussian noise (AWGN), and the ITU standards for mid-latitude disturbed and high-latitude moderate conditions. No AP information was used for these sensitivity measurements.

Table 6 - FT4 and FT8 decoding thresholds measured using simulations. In all cases, the decoder used block detection and (BP+OSD). For each channel and mode, two decoding thresholds are given. "No AP" is the threshold when no a priori information is available, and "max. AP" is the decoding threshold with the maximum amount of a priori information, at the end of a QSO when receiving RRR, 73, or RR73 from a QSO partner.

Channel	*Frequency Spread (Hz)*	*Diff. Path Delay (ms)*	*FT4 Decoding Threshold (dB), no AP*	*FT4 Decoding Threshold (dB), max. AP*	*FT8 Decoding Threshold (dB), no AP*	*FT8 Decoding Threshold (dB), max. AP*
AWGN	0.0	0.0	–17.5	–19.5	–20.8	–22.7
Mid-latitude quiet	0.1	0.5	–17.4	–19.4	–20.0	–22.4
Mid-latitude moderate	0.5	1.0	–15.8	–18.6	–18.8	–22.1
Mid-latitude disturbed	1.0	2.0	–15.2	–18.4	–18.6	–22.1
High-latitude moderate	10.0	3.0	–12.2	–17.4	–8.6	–18.9

beginning of a transmit interval. In such cases, *WSJT-X* will skip the missed part of the transmission and start sending a correctly synchronized message for the remainder. Those receiving such a truncated message will collect only a fraction of the message symbols. A common scenario for this occurrence is "late" responses to a CQ. For example, suppose that G4WJS calls CQ. He knows that responses to his call will be addressed to G4WJS. The "no AP" curve in **Figure 9** shows that without the use of AP information, responses from callers who start transmitting more than 5 seconds after the beginning of a Tx interval will not be decoded. The curve labeled "AP mycall" shows decoding probability as a function of start time when the decoder specifically looks for responses directed to G4WJS. In this case transmissions that were started up to 8 seconds late are likely to be decoded successfully.

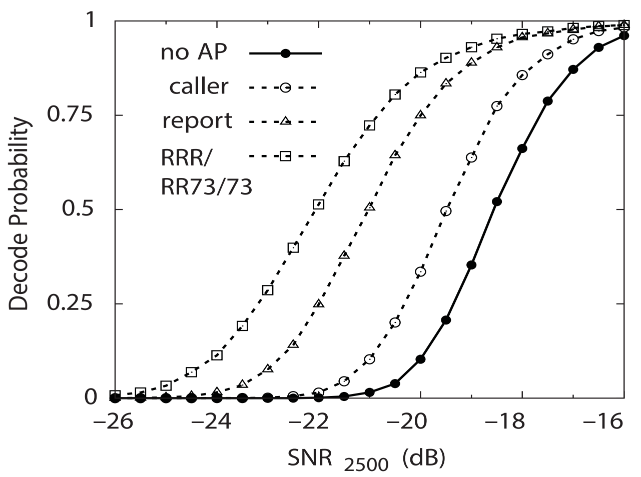


Figure 8 — Decode probability for FT8 on the mid-latitude disturbed channel using different amounts of a priori (AP) information. See text for details.

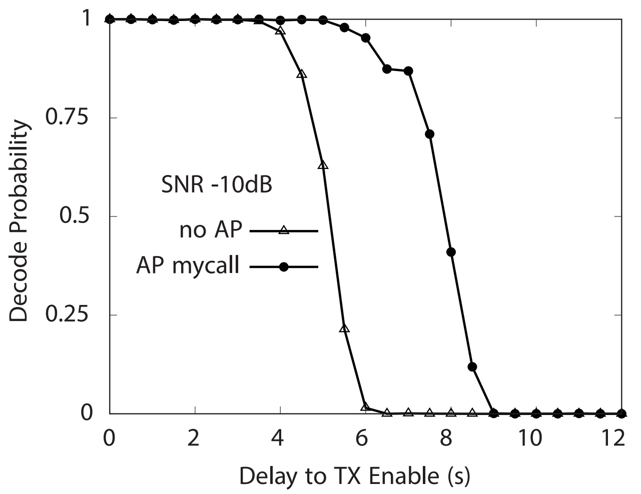


Figure 9 — Decode probability for FT8 when the transmitting station starts transmitting after the beginning of a 15 s interval. The simulation was performed for the mid-latitude moderately disturbed channel, at SNR = –10 dB. The kink in the "AP mycall" curve at 6.5 s arises because missing symbols from the Costas array in the middle of the FT8 transmission have little effect on decoding probability.

9. Concluding Remarks and Software License

This paper and the online resources found in reference **[14]** provide complete descriptions of the FT4 and FT8 protocols. In the spirit of open sharing, and to encourage other software developers who might use some of our ideas, we place this description in the public domain with the following restrictions:

- Other software implementers may use the names "FT4" and "FT8" only if they adhere to our protocol definitions for source encoding, error-correction coding, and modulation format.
- Robotic or unattended QSOs must be explicitly disallowed.
- *Multi-streaming* with waveforms and message content similar to those used in FT8 DXpedition Mode is permissible only within the guidelines specified in the *WSJT-X 2.1 User Guide* **[19]**.
- Presently unassigned message types (see **Table 1**) are reserved for future expansion and must not be assigned without our permission.
- Any implementation of these or similar protocols that allows robotic, unattended, or non-conforming multi-streaming operation shall not use the names "FT4" or "FT8" and must be made incompatible by some means, such as using different Costas arrays for synchronization.

With the exception of code contained in reference **[14]**, source code for our implementations of FT4, FT8, and MSK144 is *not* in the public domain. Rather, all code in *WSJT-X* is copyrighted and licensed under the terms of Version 3 of the GNU General Public License (GPLv3), reference **[20]**. Very briefly stated, GPLv3 guarantees end users the freedom to run, study, and modify the software, so long as the same licensing terms are extended to any new software derived from or dependent upon our source code.

We welcome any independent software implementations of FT4 and FT8, so long as they either (1) adhere to all requirements mentioned above, or (2) make no use of our source code beyond the public-domain resources mentioned above. We hope and trust that many of the innovations pioneered in *WSJT-X* and its predecessors will outlive its present developers — and will continue to advance the art of weak-signal communication by Amateur Radio for many years to come.

Appendix A. Source Encoding Details

Section 2 of this paper is a summary of how the FT4, FT8, and MSK144 protocols pioneered in *WSJT-X* compress and convey call signs, Maidenhead locators, signal reports, and certain other information in a very efficient way. **Tables 1 and 2** outline the basic source-encoding framework, with each message payload comprising a sequence of fixed-length bit fields. This Appendix completes the details needed to fully define mappings from human-readable message fragments to relevant fields in the fixed-size 77-bit message payload. For the convenience of others who might wish to implement the FT4 or FT8 protocols, some essential algorithmic parts of the definitions are presented in simple, very short Fortran programs whose source code we have placed in the public domain **[14]**.

As first steps we shall define the way standard call signs and 4-character Maidenhead locators are encoded into fields of just 28 and 15 bits, respectively. As described in the *WSJT-X 2.1 User Guide* **[19]**, a standard amateur call sign consists of a one- or two-character prefix, at least one of which must be a letter, followed by a decimal digit and a suffix of up to three letters. Within these rules, the number of possible call signs is equal to 37×36×10×27×27×27 =

262,177,560. (The numbers 27 and 37 arise because in the first and last three positions a character may be absent, or a letter, or perhaps a digit.) Since 2^{28} = 268,435,456 is a larger number, 28 bits are enough to encode any standard call sign uniquely, while leaving 6,257,896 values available for conveying other types of information. A few of these excess values are assigned to special message components such as DE, QRZ, and CQ (optionally with a modifier). A further 2^{22} = 4,194,304 are used to convey 22-bit hash codes for call signs.

Similarly, there are 180×180 = 32,400 four-digit Maidenhead grid locators. This number is less than 2^{15} = 32,768, so a grid locator can be uniquely represented in 15 bits. Some of the 368 fifteen-bit values not needed for grid locators are used to convey numerical signal reports of the form ±*nn* in the range -30 to +99 dB, or a blank, or one of the words RRR, RR73, or 73.

As a particular example, consider the bit allocations for message type 1, an essential part of almost all FT4 and FT8 QSOs. These basic messages usually contain two 28-bit call signs, an optional acknowledgment R, and a four-character locator, signal report, RRR, RR73, or 73. **Table 1** shows that this message type consists of fields with the tags `c28`, `r1`, `c28`, `r1`, `R1`, `g15`, and `i3`. The `r1` fields are normally 0; the value 1 implies a /R call sign suffix to indicate "rover" status in a North American VHF contest. The `R1` field conveys the absence (0) or presence (1) of an acknowledgment R before the locator or signal report conveyed by `g15`. In the final 77-bit information payload, bit fields are assembled in the left-to-right order specified in **Table 1**, followed by `n3` (if used; currently used only in type 0) and `i3`.

We now turn to concise descriptions of the algorithms mapping human-readable message fragments to the tagged bit fields with names listed in **Table 2**. For this purpose the public-domain Fortran programs contained in reference **[14]** are essential. To get started, download the compressed tarfile `ft4_ft8_protocols.tgz` from the **www.arrl.org/QEXfiles** web page, unpack it into a suitable directory and follow simple instructions for your operating system (Windows, Linux, or macOS) at the top of file `Makefile`. These steps will build executables for the seven programs `gen_crc14`, `free_text_to_f71`, `grid4_to_g15`, `grid6_to_g25`, `hashcodes`, `nonstd_to_c58`, and `std_call_to_c28` whose purposes are explained below.

In alphabetical order, concise definitions for the tagged bit fields listed in **Table 2** are as follows:

`c1:`	Used only in message type 4. Normally 0; value 1 denotes a message consisting of CQ and a nonstandard call sign, for example `CQ PJ4/K1ABC`.
`c28:`	Used to convey a standard call sign or one of the special message words CQ, DE, or QRZ, or a 22-bit call sign hash code. CQ may be followed by a modifier with three decimal digits or one to four letters. Each possible message fragment with CQ, DE, or QRZ is mapped to a specific integer value of `c28`, as indicated in **Table 7**. Call sign values are computed using the algorithm illustrated by program `std_call_to_c28`, and hash codes by program `hashcodes`.
`c58:`	Used to encode nonstandard call signs up to 11 characters, using the algorithm illustrated by program `nonstd_to_c58`.
`f71:`	Conveys arbitrary free text with up to 13 characters selected from the list `0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ+-./?` plus the blank character. The encoding algorithm is illustrated by program `free_text_to_f71`.
`g15:`	Encodes a 4-character Maidenhead locator, signal report, RRR, RR73, 73, or a blank message word, using the algorithm in program `grid4_to_g15`.
`g25:`	Encodes a 6-character Maidenhead locator using the algorithm in program `grid6_to_g25`.
`h10, h12, h22:`	Call sign hash codes of length 10, 12, or 22 bits. The hashing algorithm is defined in program `hashcodes`.
`k3:`	ARRL Field Day Class A-F, encoded as integer values 0 – 5.
`n4:`	In message type 0.3, maximum number of simultaneously transmitted signals minus 1. In message type 0.4, maximum number of simultaneously transmitted signals minus 17.
`p1:`	Absence (0) or presence (1) of call sign suffix /P.
`r1:`	Absence (0) or presence (1) of call sign suffix /R.
`r2:`	Value 0 encodes a blank message word; values 1 – 3 encode RRR, RR73, or 73.
`r3:`	Values 0 – 7 encode a signal report displayed as 52, 53, … 59 (message type 0.3) or 529, 539, … 599 (type 3).
`R1:`	Absence (0) or presence (1) of acknowledgment R before grid or report.
`r5:`	Values 0 – 31 convey signal reports –30, –28, … +32 dB (even numbers only).
`s11:`	Serial number, range 0 – 2047.
`s13:`	Values 0 – 7999 convey a serial number; values 8001 – 8065 encode the abbreviations for US states and Canadian provinces by means of lookup table `states_provinces.txt` included in reference **[14]**.
`S7:`	Values 0 – 83 encode abbreviations for the ARRL/RAC section names by means of lookup table `arrl_rac_sections.txt` included in reference **[14]**.
`t1:`	Absence (0) or presence (1) of TU; at the start of a type 3 message.
`t71:`	Telemetry data, up to 18 hexadecimal digits or 71 bits maximum. With 18 digits, the first digit must fall in the range 0 – 7.

We note here that for FT4 only, in order to avoid transmitting a long string of zeros when sending CQ messages, the assembled 77-bit message is bitwise exclusive-OR'ed with the following pseudo-random sequence before computing the CRC and FEC parity bits:

```
0100101001011110100010
0110110100101100001000
1010011110010101010110
11111000101
```

The receiving software applies this exclusive-OR procedure a second time, to restore the original 77-bit message.

Table 7 - Summary of mappings between message fragments and numerical values of the bit field *c28*.

Message fragment	*c28 as decimal integer*
DE	0
QRZ	1
CQ	2
CQ 000 - CQ 999	3 to 1002
CQ A - CQ Z	1004 to 1029
CQ AA - CQ ZZ	1031 to 1731
CQ AAA - CQ ZZZ	1760 to 20685
CQ AAAA - CQ ZZZZ	21443 to 532443
22-bit hash codes	2063592 + (0 to 4194303)
Standard call signs	6257896 + (0 to 268435455)

Steve Franke, K9AN, holds an Amateur Extra class license. He was first licensed in 1971 and has previously held call signs WN9IIQ and WB9IIQ. An early and abiding fascination with radio science led to a 35-year career as Professor of Electrical and Computer Engineering at the University of Illinois. Steve retired in 2019 and now holds the title Professor Emeritus. He enjoys chasing DX, playing with RF circuits and antennas, and studying HF and VHF propagation. Steve is a member of ARRL and a Fellow of the IEEE.

Bill Somerville, G4WJS, earned a Chemistry degree at the University of Bristol and has worked in computer software and hardware in a variety of industries including defense, software development, and financial services. Most recently he's a freelance consultant providing systems programming and related services to mid- to large-size software tool vendors. An active radio amateur since 1981, Bill enjoys HF and VHF bands, contest operating, and DX chasing using CW, phone, and data modes.

Joe Taylor was first licensed as KN2ITP in 1954, and has since held call signs K2ITP, WA1LXQ, W1HFV, VK2BJX and K1JT. He was Professor of Astronomy at the University of Massachusetts from 1969 to 1981 and since then Professor of Physics at Princeton University, serving there also as Dean of the Faculty for six years. He was awarded the Nobel Prize in Physics in 1993 for discovery of the first orbiting pulsar, leading to observations that established the existence of gravitational waves. After retirement he has been busy developing and enhancing digital protocols for weak-signal communication by Amateur Radio, including JT65 and WSPR. He chases DX from 160 meters through the microwave bands.

References

[1] Joe Taylor, K1JT, Steve Franke, K9AN, and Bill Somerville, G4WJS, "Work the World with *WSJT-X*: Part 1, Operating Capabilities," *QST*, Oct. 2017, pp. 30-36.
[2] Joe Taylor, K1JT, Steve Franke, K9AN, and Bill Somerville, G4WJS, "Work the World with *WSJT-X*: Part 2, Codes, Modes, and Cooperative Software Development," *QST*, Nov. 2017, pp. 34-39.
[3] "Mode Usage Evaluation: 2017 was 'the Year When Digital Modes Changed Forever'," **www.arrl.org/news/mode-usage-evaluation-2017-was-the-year-when-digital-modes-changed-forever**.
[4] Joe Taylor, K1JT, "WSJT: New Software for VHF Meteor-Scatter Communication," *QST*, Dec. 2001, pp. 36-41.
[5] Joe Taylor, K1JT, "EME with JT65," *QST*, June 2005, pp. 80–82.
[6] Joe Taylor, K1JT, "The JT65 Communications Protocol," *QEX*, Sep./Oct. 2005, p. 3-12.
[7] Rex Moncur, VK7MO, and Joe Taylor, K1JT, "Small Station EME at 10 and 24 GHz: GPS Locking, Doppler Correction, and JT4," *Dubus* 2/2013.
[8] Steven J. Franke, K9AN, and Joseph H. Taylor, K1JT, "Open Source Soft-Decision Decoder for the JT65 (63,12) Reed-Solomon Code," *QEX*, May/June 2016, pp. 8-17.
[9] Steven J. Franke, K9AN, and Joseph H. Taylor, K1JT, "The MSK144 Protocol for Meteor-Scatter Communication," *QEX*, July/Aug. 2017, pp. 8-14.
[10] Nico Palermo, IV3NWV, "Q-ary Repeat-Accumulate Codes for Weak Signals Communications," **www.eme2016.org/wp-content/uploads/2016/08/EME-2016-IV3NWV-Presentation.pdf**.
[11] Joe Taylor, K1JT, "*WSJT-X*: New Codes, Modes and Tools for Weak-Signal Communication," **www.physics.princeton.edu/pulsar/K1JT/K1JT_EME_2016_Venice.pdf**.
[12] Testing of HF Modems with Bandwidths of up to about 12 kHz Using Ionospheric Channel Simulators, *Recommendation ITU-R F.1487*, International Telecommunications Union, 2000.
[13] Ross N. Williams, "A Painless Guide to CRC Error Detection Algorithms," **https://www.zlib.net/crc_v3.txt**.
[14] **http://physics.princeton.edu/pulsar/k1jt/ft4_ft8_protocols.tgz**.
[15] Marvin K. Simon and Dariush Divsalar, "Maximum-Likelihood Block Detection of Noncoherent Continuous Phase Modulation," *IEEE Transactions on Communications*, Vol. 41, No. 1, Jan. 1993, pp. 90-98.
[16] Shu Lin and Daniel J. Costello, Jr., **Error Control Coding: Fundamentals and Applications, 2nd edition**. Pearson Prentice Hall, 2004.
[17] Yingquan Wu and Christoforos N. Hadjicostis, "Soft-Decision Decoding of Linear Block Codes Using Preprocessing and Diversification," *IEEE Transactions on Information Theory*, Vol. 53, No. 1, Jan. 2007, pp. 378-393.
[18] **https://en.wikipedia.org/wiki/UML_state_machine**.
[19] WSJT Development Group, "WSJT-X 2.1 User Guide," **www.physics.princeton.edu/pulsar/K1JT/wsjtx-doc/wsjtx-main-2.1.2.html**.
[20] GNU General Public License, GPLv3, **https://www.gnu.org/licenses/gpl-3.0.txt**. See also **https://en.wikipedia.org/wiki/GNU_General_Public_License**.