Opera Protocol Specification

Guido, PE1NNZ

Introduction

While Opera is in its experimental stages, the internal workings are hardly known. In order to address this mystery (and my personal curiosity) I have looking in the Opera symbols construction (encoding) process by comparing the output symbols for various callsign inputs. As it might have your interest too what is happening behind the scenes, here are my findings so far with Opera v1.1.9 to v1.2.8:

1. Message

First, a 51-bit message is constructed based on the operator's call-sign. The message contains in sequence:

bit 47-50	4 unused bits (set to zero)
bit 19-46	28-bit packed integer holding a compressed instance of the callsign
bit 3-18	16-bit check-sum of binary representation of bits 19-46 (this is the check-sum of the packed callsign)
bit 0-2	3-bit check-sum of binary representation of bits 19-3 (this is the check-sum of the packed call-sign and its checksum)

The 28-bit packed integer is constructed from the 6 callsign characters, where the last digit in the prefix must be aligned to the 3rd position. Each position is encoded in a certain radix and character encoding mapping:

Position	Radix	Character encoding mapping
1	37	blank(0), A-Z(1-26), 0-9(27-37)
2	36	A-Z(0-25), 0-9(26-36)
3	10	0-9(0-9)
4-6	27	blank(0), A-Z(1-26)

The check-sums are based on a CRC with polynomial $x^{16} + x^{15} + x^2 + 1$ also known as CRC-16. *Rationale for applying a 16-bit CRC is to be able to block invalid messages on the receiver site while decoding a (error corrected) message.* Note that:

a. check-sum is based on the binary representation in ASCII of the bits specified (Rationale: should be not necessary, probably a practical reason as the application can be based on string manipulations instead of binary operations);
b. a zero in the high- or low byte of the CRC result is replaced by respectively 2B (hex) and 1B (hex), Rationale: should be not necessary, unclear why this is done;

c. the high- and the low bytes are swapped when put in the message where for the 3-bit check-sum only the least significant bits are considered;

2. Scrambling
Second, a 51-bit scrambled message is constructed by means of a XOR operation with message and a pseudo-random noise vector 70ABF3680C8AB (hex). Rationale for scrambling is to reduce repetitive zeros by ensuring for enough transitions in the data independent on the end-users input, preventing degraded hamming distances for zero code words, nevertheless for the current code used it seems not to be necessary as the distance is constant for k=3.

3. Walsh-Hadamard code
Third, a Walsh-Hadamard code (k=3) is applied on the 51-bit message resulting in a 119-bit message. With k=3 the Walsh-Hadamard code maps every 3 bits of the message to a orthogonal codeword of length $2^{(k-1)}$ = 7 bits, where the Hadamard code is obtained from an inverse 8th order Hadamard matrix:

000	0000000	001	1010101	010	0110011	011	1100110
100	0001111	101	1011010	110	0111100	111	1101001

Rationale: Walsh-Hadamard code is an error-correcting code (block code) that may be used on the receiver site for forward error correction (FEC) and error detection. Doing so, relaxes the signal requirements while transmitting messages over very noise or unreliable (fading) channels, approximating the Shannon capacity model. With k=3, the minimum hamming distance of the code is $2^{(k-1)}$ =4, with this code a theoretical receiver can detect $2^{(k-2)}$-1 = 1 error and correct $2^{(k-3)}$-1 = 0 errors. The Walsh-Hadamard code is a locally decode-able code, which provides a way to recover parts of the original message with high probability, while only looking at a small fraction of the received word. As an alternative to error-correction, list-decoding may recover the original message on the receiver as long as less than 1/2 of the bits in the received word have been corrupted.

4. Interleaving
Fourth, the 119-bits message is block interleaved by writing the bits by column in a 7-by-17 matrix and reading back by rows into the message. *Rationale: interleaving makes the block encoded message less vulnerable against temporary decoding errors (fading, burst errors) by scattering of the bits over the entire message (approximately written every 7 positions).*

5. Manchester encoding
Fifth, the 119-bits message is Manchester encoded (per IEEE802.3 convention) by encoding a 1-0 transition for a 0, and a 0-1 transition for a 1, doubling the message length. Note that bit 238 and 239 are set to one and that bit 0 is omitted, resulting in a 239-bit message. The binary representation contains the symbols to be transmitted.
Rationale: With Manchester encoding a transition is encoded for every data bit so that a receiver is able to recover the clock.

6. Modulation

Sixth, in the event of an transmission event, a frequency that is not occupied is selected (that is the frequency that has the least energy just before transmission), and each of the 239 symbols are transmitted by keying the transmitter as CW on and off with a symbol rate of 0.256*n s/symbol, where n is the integer of operation mode OPn that corresponds with the Opera frequency recommendation:

OP32	137.5-137.6 kHz
OP8	137.7-137.5, 501.5-501.6, 1837.3-1837.5 kHz
OP4	501.3-501.5, 1837.5-1837.9, 3576.3-3576.5, 5290.3-5290.5, 7039.3-7039.5, 10136.3-10136.5, 14066.3-14066.5 kHz
OP2	3576.5-3576.9, 5290.5-5290.9, 7039.5-7039.9, 10136.5-10136.9, 14066.5-14066.9, 18106.3-18106.5, 21075.3-21075.5, 24926.3-24926.5, 28076.3-28076.5, 50701.3-50701.5, 70252.3-70252.5 kHz
OP1	18106.5-18106.9, 21075.5-21075.9, 24926.5-24926.9, 28076.5-28076.9, 50701.5-50701.9, 70252.5-70525.9 kHz

Discussion

On the decoder site, the inverse of above operations is done in reverse order to retrieve the callsign of a received broadcast. An interesting feature is to exploit the list-decoding potential of the Walsh-Hadamard code in combination with the CRC check-sums in the message. With list-decoding, a set of message possibilities is outputted and may be evaluated for validity. This exchanges computation power with receiver gain.

Example

Callsign = “AA1AA “

1. Message
Character Encoding Mappings: 1 0 1 1 1 0

Packed integer = (((((1) * 36 + 0) * 10 + 1) * 27 + 1) * 27 + 1) * 27 + 0 = 7106319 =
0000011011000110111100001111 (binary)

CRC-16 of Packed integer binary =
CRC-16 of 0000011011000110111100001111 =
0C1E (hex) (source: http://www.zorc.breitbandkatze.de/crc.html) →
replace zero bytes and swap CRC = 1E0C (hex) = 0001111000001100

CRC-16 of Packed integer binary + CRC-16 binary = CRC-16 of
00000110110001101111000011110001111000001100 =
6CAC (hex) → replace zero bytes and swap CRC = AC6C (hex) =
1001000101111101100100 →
last 3 bits of swapped CRC = 100

51-bit message = 000000000110110001101111000011110001111000001100100

2. Scrambling
000000000110110001101111000011110001111000001100100 XOR with Noise Vector:
111000010101011111100110110100000001100100010101011 =
111000010011101110001001110111110000011100011001111

3. Walsh-Hadamard code
Segment in 3: 111 000 010 011 101 110 001 001 110 111 110 000 011 100 011 001 111
and lookup 7-bit codeword =
1101001 0000000 0110011 1100110 1011010 0111100 1010101 1010101 0111100
1101001 0111100 0000000 1100110 0001111 1100110 1010101 1101001

4. Interleave
Write by column in 7x17 matrix:
1 1 0 1 0 0 1
0 0 0 0 0 0 0
0 1 1 0 0 1 1
1 1 0 0 1 1 0
1 0 1 1 0 1 0
0 1 1 1 1 0 0
1 0 1 0 1 0 1
1 0 1 0 1 0 1
0 1 1 1 1 0 0
1 1 0 1 0 0 1

0 1 1 1 1 0 0
0 0 0 0 0 0 0
1 1 0 0 1 1 0
0 0 0 1 1 1 1
1 1 0 0 1 1 0
1 0 1 0 1 0 1
1 1 0 1 0 0 1
Read by column =
100110110100101111011010011101010100101111101000010100011001110010010001
01111010111100011100000001110010100011010001011

5. Manchester encoding
Replacing 0 to Z and 1 by 01 and Z by 10:
Manchester encoded message =
011010010110010110011010011001010101100101100110100101011001100110011010
011001010101011001101010100110011010100101101001010110100110100110101001
100101010110011001010101101010010101101010101010100101011010011001101010
0101100110101001100101 →
Insert 11 on front and remove last bit on tail =

Opera Symbols (length 239) =
110110100101100101100110100110010101011001011001101001010110011001100110
100110010101010110011010101001100110101001011010010101101001101001101010
011001010101100110010101011010100101011010101010101001010110100110011010
10010110011010100110010