NATO STANDARD

AComP-4415

CHARACTERISTICS OF A ROBUST, NON-HOPPING, SERIAL TONE MODULATOR/DEMODULATOR FOR SEVERELY DEGRADED HF RADIO LINKS

Edition A Version 1

MARCH 2015

NORTH ATLANTIC TREATY ORGANIZATION

ALLIED COMMUNICATION PUBLICATION

Published by the
NATO STANDARDIZATION OFFICE (NSO)
© NATO/OTAN

INTENTIONALLY BLANK

NORTH ATLANTIC TREATY ORGANIZATION (NATO)

NATO STANDARDIZATION OFFICE (NSO)

NATO LETTER OF PROMULGATION

3 March 2015

1. The enclosed Allied Communication Publication AComP-4415, Edition A, Version 1 - CHARACTERISTICS OF A ROBUST, NON-HOPPING, SERIAL TONE MODULATOR / DEMODULATOR FOR SEVERELY DEGRADED HF RADIO LINKS, which has been approved by the nations in the C3B, is promulgated herewith. The agreement of nations to use this publication is recorded in STANAG 4415.

2. AComP-4415, Edition A, Version 1 is effective upon receipt.

3. No part of this publication may be reproduced, stored in a retrieval system, used commercially, adapted, or transmitted in any form or by any means, electronic, mechanical, photo-copying, recording or otherwise, without the prior permission of the publisher. With the exception of commercial sales, this does not apply to member nations and Partnership for Peace countries, or NATO commands and bodies.

4. This publication shall be handled in accordance with C-M(2002)60.

[illegible]

Edvardas MAŽEIKIS
Major General, LTUAF
Director, NATO Standardization Office

INTENTIONALLY BLANK

RESERVED FOR NATIONAL LETTER OF PROMULGATION

INTENTIONALLY BLANK

RECORD OF RESERVATIONS

CHAPTER	RECORD OF RESERVATION BY NATIONS
Note: The reservations listed on this page include only those that were recorded at time of promulgation and may not be complete. Refer to the NATO Standardization Document Database for the complete list of existing reservations.	

INTENTIONALLY BLANK

RECORD OF SPECIFIC RESERVATIONS

[nation]	[detail of reservation]
Note: The reservations listed on this page include only those that were recorded at time of promulgation and may not be complete. Refer to the NATO Standardization Document Database for the complete list of existing reservations.	

INTENTIONALLY BLANK

TABLE OF CONTENTS

INTENTIONALLY BLANK

CHAPTER 1 INTRODUCTION

1.1. INTRODUCTION

1.1.1. This document describes the transmission format, error correction coding, and modulation required to ensure interoperability between modems transmitting data over HF radio links where the user provided data rate is 75 bits per second. This 75 bits per second modem standard is required for operation in severely degraded HF channel conditions. Robustness against poor signal to noise ratios, very severe Doppler spreads and large multipath spreads is provided. The specified modem waveform incorporates a convolutional FEC combined with a Long (4.8 second) block interleaver which must be used to provide the robust performance required under these adverse channel conditions. Short and Zero interleaver settings are also defined to allow the user to trade off performance and end to end delay, under less severe channel conditions.

1.1.2. The on-air waveform specified in this document is identical to the 75 bps waveform of Mil-Std-188-110. Modems built to meet the standard specified in this document will be able to process 75 bps Mil-Std-188-110 transmissions and vice versa. However, STANAG 4415 modems are required to meet more challenging multipath delay and Doppler spread performance targets. Having Mil-Std-188-110 and STANAG 4415 on-air interoperability at 75 bps imposes the restriction that there is no distinction between the zero and short interleaver settings. This means that the users must ensure that both ends of the link have the same modem configuration setting (i.e. either short or zero interleaving) for successful communications. To provide guidance for users, zero interleaving is not recommended for robust applications or degraded channel conditions.

1.2. AIM

The aim of this agreement is to define the technical interoperability characteristics for a means of digital communications over severely degraded HF radio links at an effective bit rate of 75 bits per second.

1.3. AGREEMENT

The participating nations agree to use the characteristics contained in this STANAG for their serial tone modems for data communications over severely degraded HF Radio Links.

1.4. IMPLEMENTATION OF THE AGREEMENT

This STANAG is implemented by a nation when serial tone modems for data communications at 75 bits per second comply with the characteristics detailed in this agreement and are placed in service.

1.5. RELATED DOCUMENTS

1.5.1. STANAG 4285 Characteristics of 1200/2400/3600 Bits Per Second Single Tone Modulators/Demodulators for HF Radio Links.

1.5.2. U.S. MIL-STD 188-110 - Interoperability and Performance Standards for Data Modems

CHAPTER 2 TRANSMISSION FORMAT

2.1. GENERAL

2.1.1. Each data transmission consists of four distinct phases; the Synchronisation Pre-amble Phase; the Data Phase; the End Of Message (EOM) Phase; and the Coder - Interleaver Flush Phase. Figure 2.1 shows a block diagram view of the transmit processing.

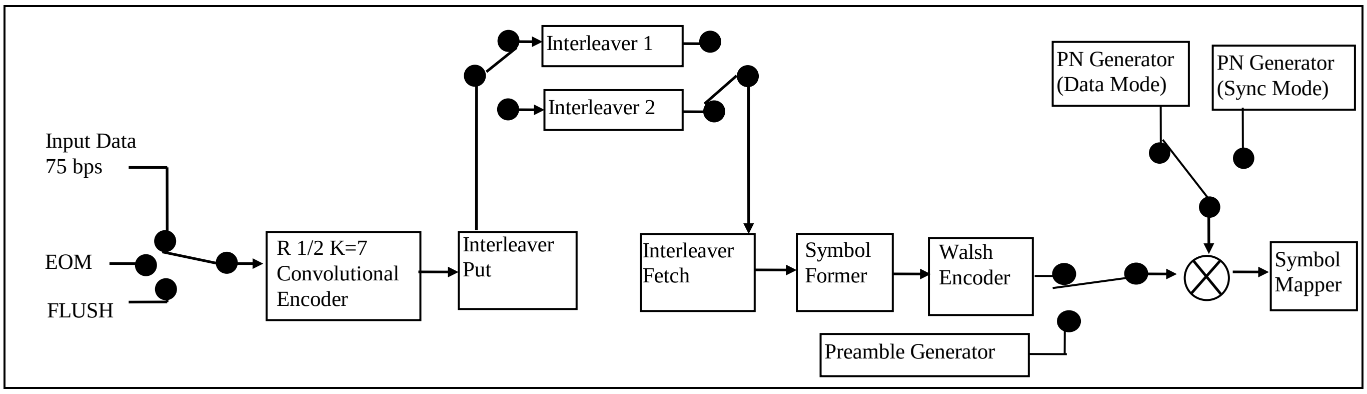

Figure 2.1 - Transmit Processing

2.1.2. All phases employ 8-ary Phase Shift Keying of a single 1800Hz sub-carrier. The modulation of this output carrier should be a constant 2400 symbols per second. The modulation phases are defined in Table 2.1.

2.1.3. The accuracy of the clock linked with the generation of the sub-carrier frequency is 1 part in 10^5, compatible with STANAG 4285.

2.1.4. The Phase Shift of the signal relative to that of the unmodulated sub-carrier may take on one of the following values.

Symbol Index	Phase Shift	In-Phase	Quadrature
0	**0**	**1.0**	**0.0**
1	$\pi/4$	$1/\sqrt{2}$	$1/\sqrt{2}$
2	$\pi/2$	**0.0**	**1.0**
3	$3\pi/4$	$-1/\sqrt{2}$	$1/\sqrt{2}$
4	π	**-1.0**	**0.0**
5	$5\pi/4$	$-1/\sqrt{2}$	$-1/\sqrt{2}$
6	$3\pi/2$	**0.0**	**-1.0**
7	$7\pi/4$	$1/\sqrt{2}$	$-1/\sqrt{2}$

Table 2.1 - 8-ary PSK signal space

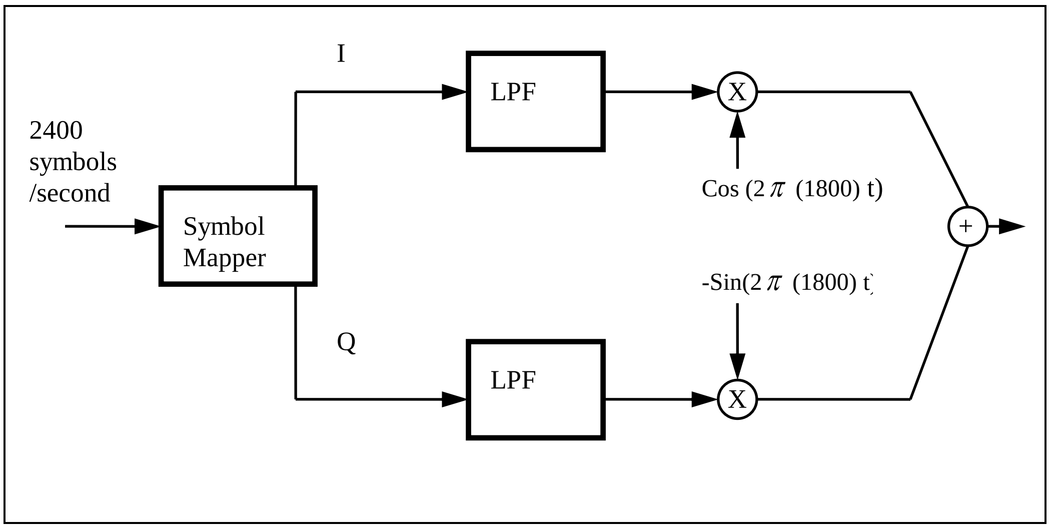

Figure 2.2 - Transmit Waveform Generation

2.1.5. The transmitted waveform is generated as illustrated in Figure 2.2. The 8-ary data stream is converted to the complex 8-PSK resulting in separate In-Phase [I] and Quadrature [Q] waveforms. These waveforms are independently filtered by equivalent low pass filters to provide spectral containment. Finally the In-phase and Quadrature waveforms are used to modulate the 1800 Hz sub-carrier.

2.2. SYNCHRONISATION PREAMBLE PHASE

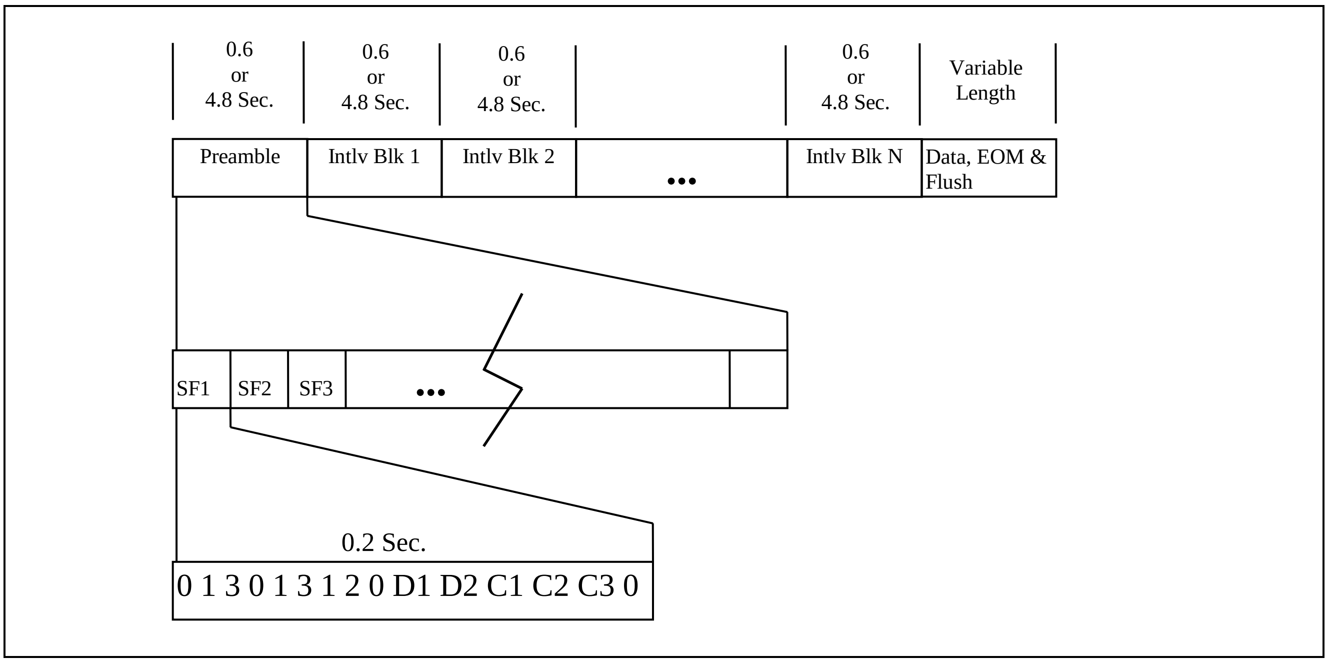

Figure 2.1.1 Transmit Structure

2.2.1. As shown in Figure 2.1.1, the preamble phase consists of the transmission of a number of preamble superframes. 24 preamble superframes are sent for the Long interleaver mode, 3 are sent in both the Short and Zero interleaver modes. Each preamble superframe consists of the transmission of 15 orthogonally modulated frames. Frames are orthogonally modulated using Walsh functions and are a total of 32 symbols in duration. This process is described in Appendix 1 of ANNEX A.

2.2.2. The duration of the preamble phase is equal to that of an interleaver block. This allows the modem to fill the interleaver block while transmitting the preamble. This ensures that transmit data is available at the end of the preamble transmission.

2.2.3. Each preamble superframe consists of the following frame sequence

0,1,3,0,1,3,1,2,0,D1,D2,C1,C2,C3,0

D1 and D2, (used to identify the data rate and interleaver setting in MIL-STD-188-110), are represented as tribit numbers and are fixed as specified in Table 2.1.1. Note that the implementation does not distinguish between Zero and Short interleaver settings. This is a modem configuration item that must be set up by the user.

Mode	D1	D2
STANAG 4415 - Zero	**7**	**5**
STANAG 4415 - Short	**7**	**5**
STANAG 4415 - Long	**5**	**5**

Table 2.1.1 - Preamble D1 D2 symbols

2.2.4. C1,C2, and C3 represent a count which is decremented for each transmission of the preamble superframe. As stated previously, this preamble superframe is repeated 24 times for Long interleaver setting and is repeated 3 times for the Zero and Short interleaver settings. The count specified by C1 C2 and C3 starts at either 23 or 2. The values of C1, C2, and C3 are represented as tribit numbers as shown in Table 2.1.2.

Count	**C1**	**C2**	**C3**
23 (010111)	**5**	**5**	**7**
22 (010110)	**5**	**5**	**6**
21 (010101)	**5**	**5**	**5**
20 (010100)	**5**	**5**	**4**
19 (010011)	**5**	**4**	**7**
18 (010010)	**5**	**4**	**6**
17 (010001)	**5**	**4**	**5**
16 (010000)	**5**	**4**	**4**
15 (001111)	**4**	**7**	**7**
14 (001110)	**4**	**7**	**6**
13 (001101)	**4**	**7**	**5**
12 (001100)	**4**	**7**	**4**
11 (001011)	**4**	**6**	**7**
10 (001010)	**4**	**6**	**6**

9 (001001)	**4**	**6**	**5**
8 (001000)	**4**	**6**	**4**
7 (000111)	**4**	**5**	**7**
6 (000110)	**4**	**5**	**6**
5 (000101)	**4**	**5**	**5**
4 (000100)	**4**	**5**	**4**
3 (000011)	**4**	**4**	**7**
2 (000010)	**4**	**4**	**6**
1 (000001)	**4**	**4**	**5**
0 (000000)	**4**	**4**	**4**

Table 2.1.2 - Preamble C1 C2 C3 symbols

2.3. DATA PHASE

2.3.1. The data phase consists of the sequential operations of convolutional encoding, interleaver put, interleaver fetch, symbol formation, and frame modulation. The processing flow is illustrated in Figure 2.1. Note that two interleaver buffers are used. While one buffer is being filled, the other buffer is being emptied.

2.3.2. Convolutional Encoder - Figure 2.2.1 displays the structure of the K=7, rate 1/2 convolutional encoder utilised for FEC. The polynomial corresponding to output b0 is $X^6 + X^4 + X^3 + X + 1$, the polynomial corresponding to output b1 is $X^6 + X^5 + X^4 + X^3 + 1$. A single input bit results in the bit pair b0, b1 being generated.

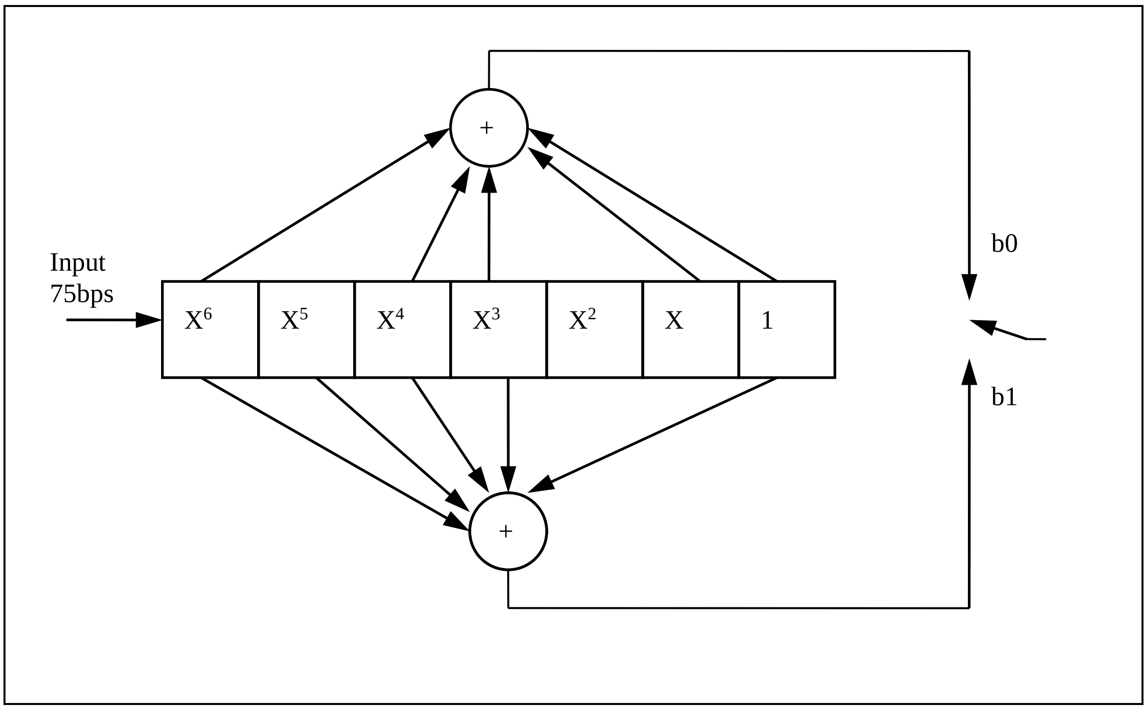

Figure 2.2.1 - Rate 1/2 K=7 Convolutional Encoder

2.3.3. Interleaver Put - The interleaver shall be a matrix block type which operates upon input bits. The matrix size shall accommodate 4.8 (Long) or 0.6 (Short) seconds of message bits . If the Zero interleaver setting is selected the interleaver will be bypassed. The dimensions of the interleaver structure are specified in the following table:

Transmit Mode	Number Of Rows	Number Of Columns
STANAG 4415 - Short	**10**	**9**
STANAG 4415 - Long	**20**	**36**

Table 2.2.1 - Interleaver Dimensions

2.3.4. Because the bits are loaded and fetched in different orders, two distinct interleaver matrices will be required. One structure will always be filling while the other is being emptied.

2.3.5. The output bits of the encoder shall be loaded into the interleaver matrix starting at column 0 as follows: The first bit (b0) is loaded into row 0, the next bit (b1) is loaded into row 7, the third bit (b0), generated from the second encoder input bit, is loaded into row 14 and the fourth bit (b1) is loaded into row 1. Thus the row location for the bits increases by 7 modulo (Number of Rows). This process continues until all rows are loaded. The load then advances to column 1 and the process is repeated until the entire matrix block is filled.

2.3.6. Interleaver fetch - The fetching sequence shall start with the first bit being taken from row 0, column 0. The location of each successive fetched bit shall be determined by incrementing the row by 1 and decrementing the column number by 7 (modulo the number of columns in the interleaver matrix) . The interleaver fetch shall continue until the row number reaches its maximum value. At this point the row number shall be reset to 0, the column number is set to be 1 larger than the value it had when the row number was last 0 and the process continued until the entire matrix is unloaded.

2.3.7. Symbol formation - pairs of bits from the interleaver fetch are mapped into 2 bit symbols as shown in Table 2.2.2.

Bits from Interleaver		**Transmitted dibit**
First Bit	**Last Bit**	
0	**0**	**00**
0	**1**	**01**
1	**0**	**11**
1	**1**	**10**

Table 2.2.2 - Symbol Formation

2.3.8. Frame Modulation - The transmit dibit, specified in Table 2.2.2, is next used to modulate a frame as detailed in ANNEX A.
2.3.9. The end of every transmitted interleaver block is identified by slightly altering the transmit dibit to a tribit by adding 4. This results in an alternate set of Walsh functions being used for the frame modulation. This is done to identify the end of a interleaver block; allowing synchronisation on the data portion of the waveform even if the preamble is missed.

2.4. END OF MESSAGE PHASE

2.4.1. After all the user message bits have been processed, as outlined in the data mode, the 32 bit End Of Message sequence is processed by the encoder and interleaver. The 32 bit EOM sequence is defined as:

0,1,0,0,1,0,1,1,0,1,1,0,0,1,0,1,1,0,1,0,0,1,0,1,1,0,1,1,0,0,1,0

2.4.2. The bits above are processed left to right, and they are treated as input data to the system (see Figure 2.1).

2.5. CODER AND INTERLEAVER FLUSH PHASE

After the 32 bit EOM sequence has been processed 144 0 bits are injected into the encoder / interleaver. These 144 bits allow the encoder to be brought to a known state and provide additional bits to flush the decoder. After the 144 0’s have been processed additional zero’s are injected into the encoder until the current interleaver block is filled so that it can be emptied and transmitted.

CHAPTER 3 MINIMUM REQUIRED PERFORMANCE

3.1. MINIMUM REQUIRED PERFORMANCE

3.1.1. The performance specified in the following paragraphs is required when the modem is operating in the long interleaver mode. The HF simulator used shall be in accordance with CCIR Report 549-3. Doppler spread shall be Gaussian and specified as a 2 Sigma power bandwidth. Signal and noise powers shall be measured in a 3 kHz bandwidth.

3.1.2. Single Path, non-fading - The modem shall achieve a BER<10^{-3} at -9.00 dB SNR (3kHz) in an additive white Gaussian noise environment.

3.1.3. Dual Path, Multipath delay = 10.0ms. - The modem shall achieve a BER < 10^{-4} at the following SNRs (3kHz).

Doppler Spread (both paths) (Hz)	Required SNR[dB] to achieve 10^{-4}BER
0.5	0.0
1.0	-1.0
2.0	-1.0
5.0	-1.0
10.0	-1.0
20.0	-1.0
30.0	-1.0
40.0	-0.5
50.0	0.0

Table 3.1 - Fading Multipath Performance

3.1.4. Delay Spread Tolerance - The modem shall be capable of achieving synchronisation and providing BER of less than 10^{-5} for multipath delay spreads up to 10 milliseconds in a 0 dB SNR channel with Doppler spreads of 2 Hz and 20 Hz.

3.1.5. Interference Tolerance - Table 3.2 specifies Signal to Interference Ratio (SIR) that shall be accommodated by the modem while maintaining a BER of 10^{-4} for several different types of interference. In order to obtain the stated performance it may be necessary to implement excision filters in the demodulator.

Interference Type 1 path non-fading SNR = +10dB	Details	SIR required to achieve 10^{-4}BER
Swept CW	Triangular sweep 0-3000 Hz, 150 Hz / second	≥ -40 dB
FSK - WS 75bps	1575, 2425 Hz	≥ -40 dB

Table 3.2 - Interference Performance

3.1.6. Doppler Shift Tolerance - The modem shall be capable of achieving synchronisation and providing a BER of less than 10^{-5} for Doppler shifts up to +/- 75 Hertz on a single fading path with 0 dB SNR and 2 Hertz Doppler spread.

3.1.7. Doppler Sweep Tolerance - The modem shall be capable of achieving synchronisation and providing a BER of less than 10^{-5} for a continuous linear Doppler sweep between +/- 75 Hz at a rate of 3.5 Hz/s for a single non-fading path with a 0 dB SNR.

ANNEX A ORTHOGONAL MODULATION OF FRAMES

1. All data is transmitted as frames. Each frame is completely defined by specifying the underlying 32 symbol 8-PSK base pn sequence and the orthogonal modulation index 0-7. The base pn sequence is used as a scrambling sequence so that the transmitted waveform always appears noise-like in nature in-dependent of the actual user transmitted data. The modulation is accomplished by the modulo 8 addition of selected Walsh functions and the underlying 8-PSK base pn sequence.

2. The following base pn sequence is utilised in the generation of preamble synchronisation frames:

7,4,3,0,5,1,5,0,2,2,1,1,5,7,4,3,5,0,2,6,2,1,6,2,0,0,5,0,5,2,6,6

where each integer 0-7 corresponds to a phase of 8-PSK as specified in Table 2.1.

3. Data frames are generated from the following 160 symbol sequence, taken 32 at a time, repeated after every 5 frames:

0,2,4,3,3,6,4,5,7,6,7,0,5,5,4,3,5,4,3,7,0,7,6,2,6,2,4,6,7,2,4,7,
5,5,7,0,7,3,3,3,7,3,3,1,4,2,3,7,0,2,7,7,3,5,1,0,1,4,0,5,0,0,0,0,
7,5,1,4,5,4,2,0,6,1,4,7,5,0,1,0,3,0,3,1,3,5,1,2,5,0,1,7,1,4,6,0,
2,3,3,4,2,5,2,5,4,5,7,3,1,0,1,6,4,1,1,2,1,4,1,5,4,2,7,4,5,1,6,4,
6,3,6,4,5,0,3,6,4,0,1,6,3,3,5,7,0,5,7,7,2,5,2,7,7,4,7,5,5,0,5,6,

where each integer 0-7 corresponds to a phase of 8-PSK as specified in Table 2.1.

4. Table A-1.1 specifies the orthogonal Walsh functions used in the frame modulation. The first set of four are used for all preamble frames except D1, D2, C1, C2, and C3 and for all data frames except the last frame of each interleaver block. The second set, represented by the input tribit, are used for modulation of the preamble frames D1, D2, C1, C2, C3 and for the last data frame of each interleaver block.

Orthogonal Modulation Index	Walsh Function
Input dibit	
0 (00)	**0 0 0 0 0 0 0 0**
1 (01)	**0 1 0 1 0 1 0 1**
2 (10)	**0 0 1 1 0 0 1 1**
3 (11)	**0 1 1 0 0 1 1 0**
Input tribit	
4 (100)	**0 0 0 0 1 1 1 1**
5 (101)	**0 1 0 1 1 0 1 0**
6 (110)	**0 0 1 1 1 1 0 0**
7 (111)	**0 1 1 0 1 0 0 1**

Table A-1.1 - Orthogonal Modulation

5. The frame modulation is accomplished by taking the Walsh function specified by the orthogonal modulation index and repeating it 4 times. This results in a 32 element sequence which is then used to modulate the specified base pn sequence. This modulation is accomplished by rotating each 8-ary PSK symbol of the base pn sequence by 180 degrees if the corresponding Walsh function element is a 1, and leaving it unaltered if the Walsh element is a 0.

6. Table A-1.1 shows Walsh Functions for Input “tribits”. These functions represent an alternate set of Walsh Functions that are used for the last frame of each interleaver block. This operation is used to support synchronisation on data.

7. EXAMPLE#1: Frame Modulation of the second frame of the preamble superframe.

Here, the frame data is specified as 1 and the base pn sequence is as defined in 1(b) above.

Walsh Index	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
Base PN	7,4,3,0,5,1,5,0,2,2,1,1,5,7,4,3,5,0,2,6,2,1,6,2,0,0,5,0,5,2,6,6
Transmitted	7,0,3,4,5,5,5,4,2,6,1,5,5,3,4,7,5,4,2,2,2,5,6,6,0,4,5,4,5,6,6,2

8. EXAMPLE#2: Frame Modulation of a standard data frame using the 3rd base pn sequence.

Assume the dibit, as specified in Table A-1.1 is 3.

Walsh Index	0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0
Base PN	7,5,1,4,5,4,2,0,6,1,4,7,5,0,1,0,3,0,3,1,3,5,1,2,5,0,1,7,1,4,6,0,

Transmitted 7,1,5,4,5,0,6,0,6,5,0,7,5,4,5,0,3,4,7,1,3,1,5,2,5,4,5,7,1,0,2,0,

9. EXAMPLE#3: Frame Modulation of the last data frame of the interleaver block using the 1st base pn sequence.

Assume the dibit, as specified in Table A-1.1 is 3. This results in the tribit 7 being used.

Walsh Index	0 1 1 0 1 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0 1 0 0 1
Base PN	0,2,4,3,3,6,4,5,7,6,7,0,5,5,4,3,5,4,3,7,0,7,6,2,6,2,4,6,7,2,4,7,

Transmitted 0,6,0,3,7,6,4,1,7,2,3,0,1,5,4,7,5,0,7,7,4,7,6,6,6,6,0,6,3,2,4,3,

ANNEX B SPECIFICATIONS OF ASSOCIATED COMMUNICATIONS EQUIPMENT

1. The transmitter and receiver specifications must be in accordance with Appendix 1 to Annex A to STANAG 4285 for the robust serial tone modulator/ demodulator to perform adequately.

ANNEX C MODEM DIALOGUES AND INTERFACES WITH TRANSMITTER, RECEIVER AND DATA TERMINALS (Information)

1. Modem dialogues and interfaces with transmitter, receiver, and data terminals are described in Appendix 2 of Annex A to STANAG 4285. Additional information on this subject is available in Annex F to STANAG 4481. It should be noted that baud and frame rates specified in this STANAG differ from those of STANAG 4285 and, as a consequence, adjustment must be made to clock signals for “frame transmission” and “frame reception”.

ANNEX D MODULATOR/DEMODULATOR FILTERING (Information)

1. The forming filters used by the modulator, depicted in Figure 2.1, and the demodulator should meet the following criteria.

(a) The reception filter should be matched to the transmission filter (to maximise the signal-to-noise ratio in the demodulator).

(b) Putting the transmission and reception filters in series should form a filter minimising inter-symbol interference at the demodulator. A root-raised cosine filter will satisfy the necessary requirements.

ANNEX E EXAMPLE MODULATOR - C LANGUAGE IMPLEMENTATION

```
/*************************************************************************


  This program implements the MIL_STD-188-110 75bps Modulator.
  It supports all interleaver modes;  zero, short, and long.
  The output is generated as an ASCII file containing the transmitted 8-PSK
  symbols as denoted by the integers 0-7, which correspond to the 8-PSK
  signalling phases of 0, 45, 90, 135, 180, 225, 270 and 315 degrees.

*************************************************************************/

#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <math.h>



float pi = M_PI;

int baud,interleaver;
int baud_ind, ilv_ind;
long i,numtxbits;
int fullflag;
int walsh_data;

float tx_buffer[500];

/*************************************************************************
***************    Modem Transmit Data Structures *************************
*************************************************************************/

/* Base PN sequence for Data Mode */
int dataPN[5][32] = {   {0,2,4,3,3,6,4,5,7,6,7,0,5,5,4,3,
                    5,4,3,7,0,7,6,2,6,2,4,6,7,2,4,7},
                    {5,5,7,0,7,3,3,3,7,3,3,1,4,2,3,7,
                    0,2,7,7,3,5,1,0,1,4,0,5,0,0,0,0},
                    {7,5,1,4,5,4,2,0,6,1,4,7,5,0,1,0,
                    3,0,3,1,3,5,1,2,5,0,1,7,1,4,6,0},
                    {2,3,3,4,2,5,2,5,4,5,7,3,1,0,1,6,
                    4,1,1,2,1,4,1,5,4,2,7,4,5,1,6,4},
                    {6,3,6,4,5,0,3,6,4,0,1,6,3,3,5,7,
                    0,5,7,7,2,5,2,7,7,4,7,5,5,0,5,6}};

/* Random 63 bit Data sequence */
int pn63[63] ={0,0,0,0,1,1,0,0,0,1,0,1,0,0,1,1,
           1,1,0,1,0,0,0,1,1,1,0,0,1,0,0,1,
           0,1,1,0,1,1,1,0,1,1,0,0,1,1,0,1,
           0,1,0,1,1,1,1,1,1,0,0,0,0,0,1};

/* Base PN sequence for Preamble mode */
int preamblePN[32] = {7,4,3,0,5,1,5,0,2,2,1,1,5,7,4,3,
                  5,0,2,6,2,1,6,2,0,0,5,0,5,2,6,6};
```

```
struct modem_tx_state_S {
   enum tx_mode_E { NULL_TX,PREAMBLE,DATA, EOM,FLUSH } tx_mode;
   enum int_deint_E { ZERO_TX,    SHORT_TX, LONG_TX  } int_deint;
};
struct modem_tx_state_S modem_tx_state = { 0,1 };


/*  Preamble data structure */
struct preamble_S {
   int synch_frame[15];
   int sym_cnt;
   int epoch_cnt;
};
struct preamble_S preamb = { {0,1,3,0,1,3,1,2,0,0,0,0,0,0,0},
                 0,0};

int preDone;

/*  8-ary Walsh function modulation array  */
int mm_walsh[8][8] = { {0,0,0,0,0,0,0,0},
                    {0,4,0,4,0,4,0,4},
                    {0,0,4,4,0,0,4,4},
                    {0,4,4,0,0,4,4,0},
                    {0,0,0,0,4,4,4,4},
                    {0,4,0,4,4,0,4,0},
                    {0,0,4,4,4,4,0,0},
                    {0,4,4,0,4,0,0,4}};

/*  End Of Message  sequence  */
eomseq[32] = {0,1,0,0,1,0,1,1,0,1,1,0,0,1,0,1,
             1,0,1,0,0,1,0,1,1,0,1,1,0,0,1,0};

/*  Gray Encoder array -  used in 75 bps */
int mgd[4] = { 0,1,3,2};


/*  Baud and interleaver specific transmit parameters  */
struct tx_param_S  {
   int d1;
   int d2;
   int pre_cnt;
   int bits_per_sym;

   };

struct tx_param_S tx[1][3] = { 7, 5, 2, 2 ,
                           7, 5, 2, 2 ,
                           5, 5, 23, 2};

/*************************************************************************
       Encoder Data Structures
*************************************************************************/
int encode_state = 0;
int encoded_bit[2];

int encoder_tab[64] = {0,1,3,2,3,2,0,1,0,1,3,2,3,2,0,1,
                2,3,1,0,1,0,2,3,2,3,1,0,1,0,2,3,
                3,2,0,1,0,1,3,2,3,2,0,1,0,1,3,2,
                1,0,2,3,2,3,1,0,1,0,2,3,2,3,1,0};


/***************************************************************************/
/**             Interleaver Data Structures                   **/
/***************************************************************************/

int imem[40*576];

struct ilv_param_S  {
   int num_rows;
   int num_cols;
   int size;
   int symbols_per_block;
   int put_row_b;
   int get_col_b;
   };

struct ilv_var_S {
   int put_pos;
   int get_pos;
   int put_row_cnt;
   int put_col_cnt;
   int get_row_cnt;
   int get_col_cnt;
   int old_get_col_cnt;
};

struct ilv_var_S ilv_var = { 0,0,0,0,0,0,0};

struct ilv_param_S ilv[1][3] = {
           /* 75 Z */   10,   9,  90, 45, 1,  0,
           /* 75 S */   10,   9,  90, 45, 7, -7,
           /* 75 L */   20,  36, 720, 360, 7, -7
             };


/*************************************************************************
      Function Declaration
*************************************************************************/

void transmit_init( void );
int  preamble(void);
void modulator(int walsh, int xmit_sym_cnt , float *out_P);
int  interleav_put( int inbit);
int  interleav_get(void);
void encode( int input_bit );
void interleaver_process75(void);

/************************************************************************
      Start Of Main Program
************************************************************************/

/*  The main program is implemented as a series of sections dealing with
    PREAMBLE, DATA, EOM, and FLUSH phases of the transmit waveform.
    A single interleaver block is employed,  this is filled
    with encoded data,  once filled the entire interleaver block is
    converted to transmit waveform.  This approach, while acceptable for
    a non real-time implementation should be altered to be frame based for
    real-time operation.   */

FILE *pskout;

void main()
{
char intstr[2];

   printf("MIL-STD 188-110 - 75bps Transmit Model\n\n");

  /*  Initialization here  */

   modem_tx_state.tx_mode = NULL_TX;

   pskout = fopen("pskout","w");


   /* NULL_TX */

   baud = 75;
   baud_ind =  0;

   printf("Enter Interleaver (L)ong (S)hort (Z)ero ");
   scanf("%s", &intstr);

   if( (intstr[0] == 'L') || (intstr[0] == 'l'))
      modem_tx_state.int_deint = LONG_TX;

   if( (intstr[0] == 'S') || (intstr[0] == 's'))
      modem_tx_state.int_deint = SHORT_TX;

   if( (intstr[0] == 'Z') || (intstr[0] == 'z'))
      modem_tx_state.int_deint = ZERO_TX;
   ilv_ind = modem_tx_state.int_deint;

   printf("Enter number of bits to transmit ");
   scanf("%ld",&numtxbits);

   transmit_init( );

   modem_tx_state.tx_mode = PREAMBLE;


/***********************************************************************/
      /* PREAMBLE*/
/***********************************************************************/
   /*  This section utilizes the preamble state machine function to generate
   all transmitted symbols of the preamble */

   printf("Generating Preamble\n");
   while ( modem_tx_state.tx_mode == PREAMBLE )
   {
      preDone = preamble();
      modulator(walsh_data, 5 , tx_buffer);
    if( preDone == 1 ) modem_tx_state.tx_mode = DATA;
  }

/***********************************************************************/
  /* DATA */
/***********************************************************************/

  /*  This section generates the user data portion of the waveform.  In this
  example the data is taken from the pn63 array which stores the 63 bit ML
  PN sequence used by many Bit Error Rate Test Sets. */

  printf("Generating data waveform\n");
   for(i=0;i<numtxbits;i++)
  {
    encode( pn63[i%63] );

    interleav_put( encoded_bit[0] );
    fullflag = interleav_put( encoded_bit[1] );

    if( fullflag == 1)
    {
      printf("interleaver full at %ld\n", i);
      interleaver_process75();
    }

  }

/***********************************************************************/
  /*   EOM */
/***********************************************************************/

  /* This section transmits the 32 bit End Of Message Sequence  */

  printf("Generating EOM\n");
  for( i=0;i<32;i++)
  {
    encode( eomseq[i] );
    interleav_put( encoded_bit[0] );
    fullflag = interleav_put( encoded_bit[1] );




    if( fullflag == 1)
    {
      printf("interleaver full at %ld\n", i);
      interleaver_process75();
    }
  }

/***********************************************************************/
    /* FLUSH */
/***********************************************************************/


  /* This section transmits the 144 '0' flush bits */
```

```
   printf("Generating Flush bits\n");
   for( i=0;i<144;i++)
   {
      encode( 0 );
      interleav_put( encoded_bit[0] );
      fullflag = interleav_put( encoded_bit[1] );
      if( fullflag  == 1)
      {
         printf("interleaver full \n");
         interleaver_process75();
      }

   }

   /* This  section continues to transmit '0' until the current interleaver
   block is full */

   while( fullflag == 0 )
   {
      encode( 0 );
      interleav_put( encoded_bit[0] );
      fullflag = interleav_put( encoded_bit[1] );
      if ( fullflag == 1 )
      {
          printf("interleaver full \n");
          interleaver_process75();
      }
   }

   fclose(pskout);


/**************************************************************************/
void transmit_init( ) {
/**************************************************************************/

/*  This function initializes the preamble symbols which convey the selected
    baud rate and interleaver setting and also initializes the number of
    preamble superframe transmissions.  */


   preamb.synch_frame[9] = tx[baud_ind][ilv_ind].d1;
   preamb.synch_frame[10] = tx[baud_ind][ilv_ind].d2;
   preamb.sym_cnt=0;
   preamb.epoch_cnt=tx[baud_ind][ilv_ind].pre_cnt;


}



/**************************************************************************/
int preamble( void)
/**************************************************************************/

/*  This function implements the preamble state machine.  The next preamble
```

```
   symbol to transmit is calculated and the preamble count symbols are
   recalculated each preamble epoch time.  */
{
int done=0;

  walsh_data =  preamb.synch_frame[preamb.sym_cnt];
  printf("%d ",preamb.synch_frame[preamb.sym_cnt]);

  preamb.sym_cnt++;

  /*  Calculate the preamble count symbols  */
  if( preamb.sym_cnt == 11 )
  {
    preamb.synch_frame[11] = ((preamb.epoch_cnt>>4) & 0x03) +4;
    preamb.synch_frame[12] = ((preamb.epoch_cnt>>2) & 0x03) +4;
    preamb.synch_frame[13] = ((preamb.epoch_cnt) & 0x03) +4;
  }
  /* If all 15 symbols of the preamble have been sent set up for
        the next set  */

  if( preamb.sym_cnt == 15)
  {
    preamb.sym_cnt = 0;
    preamb.epoch_cnt--;

    printf("\n");

    if(preamb.epoch_cnt < 0 )
    {
      done = 1;

    }
  }
  return ( done );
}

/*****************************************************************************/
void modulator(int walsh, int xmit_sym_cnt , float *out_P)
/*****************************************************************************/
/*
   This function performs the Walsh function modulation.  The modulation is
   achieved by XORing the scramble sequence with the Walsh function,
   represented as 0's and 4's.
*/
{

int i,mod_pn[32];

  for(i=0;i<32;i++)
  {
    if (modem_tx_state.tx_mode == PREAMBLE )
      mod_pn[i] = preamblePN[i] ^ mm_walsh[walsh][i%8];
    else
      mod_pn[i] = dataPN[xmit_sym_cnt][i] ^ mm_walsh[walsh][i%8];
    fprintf(pskout,"%d\n",mod_pn[i]);
  }

}
```

```
/*************************************************************************/
void encode( int  input_bit )
/*************************************************************************/

/*  This function implements the Rate 1/2, k = 7 convolutional encoder by means
    of the encoder_tab look up table.
*/

{

int temp;

   if ( input_bit){
      temp = encoder_tab[ encode_state ] ^ 0x03;
   }
   else{
      temp = encoder_tab[ encode_state ];
   }

   encode_state = ( (encode_state<<1) + input_bit ) & 0x3F;

   encoded_bit[0] = (temp>>1)&0x01;
   encoded_bit[1] = temp&0x01;
}


/**********************************************************************/
int interleav_put( int inbit) {
/**********************************************************************/

/*  This function loads the passed input bit into the interleaver structure
    and modifies the address variables.
*/

   imem[ilv_var.put_pos]=inbit;
   ilv_var.put_pos =  (ilv_var.put_pos +ilv[baud_ind][ilv_ind].put_row_b
           *ilv[baud_ind][ilv_ind].num_cols) % ilv[baud_ind][ilv_ind].size;
   ilv_var.put_row_cnt++;
   if( ilv_var.put_row_cnt == ilv[baud_ind][ilv_ind].num_rows)
   {
      ilv_var.put_row_cnt=0;
      ilv_var.put_pos=ilv_var.put_col_cnt+1;
      ilv_var.put_col_cnt++;
      if( ilv_var.put_col_cnt == ilv[baud_ind][ilv_ind].num_cols)
      {
         ilv_var.put_col_cnt=0;
         ilv_var.put_row_cnt=0;
         ilv_var.put_pos=0;
         return(1);
      }
   }
   return(0);
}
```

```
/**************************************************************************/
int interleav_get() {
/**************************************************************************/

/*  This function fetches the next bit out of the interleaver structure
    and modifies the address variables.
*/

int outbit;

   outbit = imem[ilv_var.get_pos];
   ilv_var.get_row_cnt++;
   ilv_var.get_col_cnt += ilv[baud_ind][ilv_ind].get_col_b;
   if(ilv_var.get_col_cnt <0 )
      ilv_var.get_col_cnt += ilv[baud_ind][ilv_ind].num_cols;

   if ( ilv_var.get_row_cnt == ilv[baud_ind][ilv_ind].num_rows )
   {
      ilv_var.get_row_cnt =0;
      ilv_var.get_col_cnt = ilv_var.old_get_col_cnt+1;
      if( ilv_var.get_col_cnt == ilv[baud_ind][ilv_ind].num_cols)
      {
         ilv_var.get_col_cnt=0;
      }

      ilv_var.old_get_col_cnt = ilv_var.get_col_cnt;
   }
   ilv_var.get_pos = ilv_var.get_row_cnt*ilv[baud_ind][ilv_ind].num_cols
                  +ilv_var.get_col_cnt;

   if( ilv_var.get_pos == 0 ) printf("Interleaver emptied\n");

   return( outbit);
}


/**************************************************************************/
void interleaver_process75() {
/**************************************************************************/

/*  This function processes a filled interleaver block  */

int j,jj;
int a;
static mod_cntr=0;

    for(j=0;j<ilv[baud_ind][ilv_ind].symbols_per_block;j++){
       a=0;
       for(jj=0;jj<2;jj++)
       {
          a=(a<<1)+interleav_get();
       }


       a=mgd[a];
       if ((modem_tx_state.int_deint == ZERO_TX) && (mod_cntr == 44 )) a +=4;
       if ((modem_tx_state.int_deint == SHORT_TX) && (mod_cntr == 44 )) a +=4;
       if ((modem_tx_state.int_deint == LONG_TX) && (mod_cntr == 359 )) a +=4;
```

```
        modulator(a, mod_cntr % 5 , tx_buffer);

        mod_cntr++;

      }

      mod_cntr = 0;
}
```

INTENTIONALLY BLANK

AComP-4415 (A)(1)