DVB-SH FORWARD ERROR CORRECTION IMPLEMENTATION IN MATLAB

Ondřej Hüttl, Tomáš Kratochvíl

Department of Radio Electronics, Brno University of Technology

Purkyňova 118, 612 00 BRNO

Abstract

The paper deals with the Matlab implementation of DVB-SH (Digital Video Broadcasting – Satellite Handheld) channel encoding presented by Forward Error Correction (FEC) encoding using Turbo Code. DVB-SH use Turbo encoding process defined by 3GPP2 (The Third Generation Partnership Project 2). The same Turbo encoding process is used in e.g. CDMA2000 Spread Spectrum Systems 3GPP2 Standard. Implemented FEC encoder follows European Standard ETSI EN 302 583 and consists of two Recursive Systematic Convolutional (RSC) encoders connected in parallel and Turbo Interleaver, preceding the second RSC encoder. Each of the encoders produces an output of three bits X, Y_0 and Y_1, respectively X´, $Y´_0$ and $Y´_1$. Encoders are followed by Puncturing.

1 Introduction

The DVB-SH (Digital Video Broadcasting – Satellite Handheld) is ETSI EN 302 583 standard [1] of European digital television using hybrid satellite and terrestrial transmission to mobile receivers. Standard specifies the "Framing structure, channel coding and modulation for Satellite services to handheld devices below 3GHz". Standard is derived from the DVB-T [2], DVB-H [3] and DVB-S2 [4] standards and includes two transmission modes – an OFDM (Orthogonal Frequency Division Multiplexing) mode based on DVB-T/H and a TDM (Time Domain Multiplex) based on DVB-S2.

2 Transmission System Architecture

The DVB-SH transmission system includes OFDM and TDM modulation possibilities for the satellite path. Channel coding is composed from processes common for both modes followed by processes dedicated to specific mode. The following processes are applied to data stream [1]:

Common for both modes:

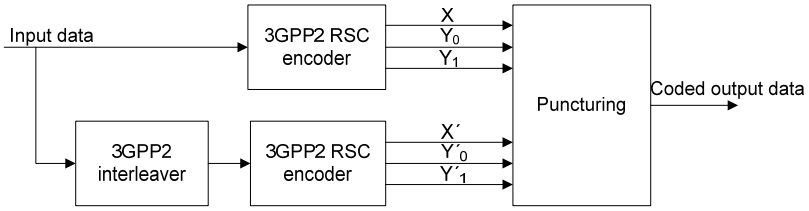
- Mode adaptation: CRC-16 and insertion of the Encapsulation Frame Header
- Stream adaptation: padding and scrambling of the Encapsulation Frame
- Forward Error Correction (FEC) encoding using 3GPP2 [5] turbo code
- Bit-wise interleaving applying on a FEC block. The latter is meanwhile shortened to comply with the modulation frame structure of OFDM and TDM
- Convolutional time interleaving and framing

TDM mode:

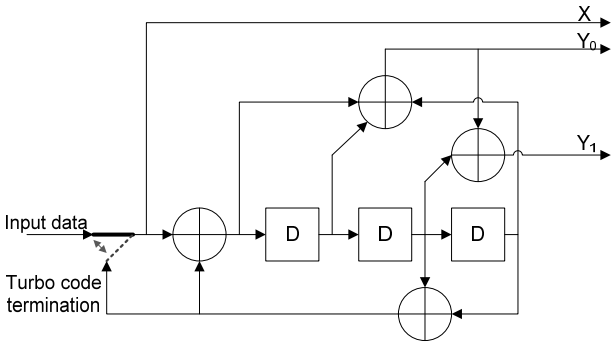
- Bit mapping to the constellation
- TDM physical layer framing
- Pilots insertion and scrambling
- Pulse shaping and quadrature modulation

OFDM mode:

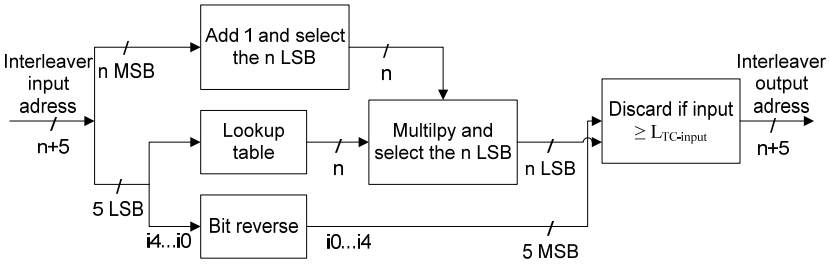
- Symbol interleaver
- Bit mapping to the constellation
- OFDM framing with pilots and TPS insertion



a) 3GPP2 Turbo code encoder



b) 3GPP2 RSC encoder



c) 3GPP2 interleaver address calculation

Figure 1: Forward Error Correction using 3GPP2 Turbo Code encoder.

3 Forward Error Correction implementation in Matlab

The DVB-SH Forward Error Correction encoding uses Turbo Code process defined by 3GPP2 with additional Code Rates (*CR*) [5]. Turbo encoder is a Block encoder working with input sequences of length $L_{\text{TC-input}}$ bits. $L_{\text{TC-input}}$ can be 1 146 of signaling bits or 12 282 data payload bits.

Following section describes possible implementation of FEC encoding for DVB-SH in accordance with European Standard ETSI EN 302 583 for digital satellite/terrestrial television. For more information please refer to [1].

3.1 Structure of 3GPP2 Turbo encoder for DVB-SH

Turbo encoder consists of two identical Recursive Systematic Convolutional (RSC) encoders connected in parallel and Turbo Interleaver, preceding the second RSC encoder. Each of the encoders produces an output of three bits X, Y_0 and Y_1, respectively X', Y'_0 and Y'_1. Encoded outputs are ended by $L_{\text{TailBits}} = 6$ Encoder Tail Bits. Turbo encoder results in encoded block of $6 \cdot (L_{\text{TC-input}} + L_{\text{TailBits}})$ bits, where 6 means X, Y_0, Y_1, X', Y'_0 and Y'_1 outputs. Encoders are followed by Puncturing with defined *CR* and the overall output block size is given as $(L_{\text{TC-input}} + L_{\text{TailBits}}) / CR$ bits.

The structure of Turbo encoder for DVB-SH is shown in the Figure 1a).

3.2 RSC encoder

RSC encoder is composed by 3 shift registers and 3 XOR (Exclusive OR) adders. Shift registers are initially set to zero states. Output bits X, Y_0 and Y_1 of the encoder are clocked $L_{\text{TC-input}}$ times with the switch in up position and Tail Bits are appended to the end of encoded sequence by clocking of the registers 6 times (L_{TailBits}). Output X contains systematic bits, Y outputs contain parity bits.

The structure of 3GPP2 RSC encoder for DVB-SH Turbo encoder is shown in the Figure 1b).

Following Matlab code performs RSC encoding of one block of $L_{\text{TC-input}}$ input data payload bits according to chapters 5.3.1 and 5.3.2 of [1]:

```
% RSC encoder
LTCinp = 12282;     %length of bits sequence encoded by the turbo encoder
input_sequence = randi([0 1],1,LTCinp);       %random input bits sequence

% Initialization
X = [];                                 %systematic bits
Y0 = [];                                %parity bits
Y1 = [];                                %parity bits
D1 = (0); D2 = (0); D3 = (0);           %Shift registers
XOR1 = [];                              %XOR1 left middle
XOR2 = [];                              %XOR2 middle top
XOR3 = [];                              %XOR3 right middle
XOR4 = xor(D2, D3);                     %XOR4 right bottom

% RSC encoding
for cl = 1:LTCinp                       %clock the registers LTCinp times
    X(cl) = input_sequence(cl);         %systematic bit written
    XOR1 = xor(X(cl),XOR4);             %XOR operations performed
    XOR2 = xor(xor(XOR1,D1),D3);
    XOR3 = xor(D2,XOR2);
    XOR4 = xor(D2,D3);
    Y0(cl) = XOR2;                      %parity bits written
    Y1(cl) = XOR3;
    D3 = D2;                            %shift of registers content
    D2 = D1;
    D1 = XOR1;
end

% Tail symbols
for tl = 1:6                            %clock the registers 6 times for tail bits
    Xtail(tl) = XOR4;                   %systematic tail bit written
    XOR1 = xor(XOR4,XOR4);              %XOR operations performed
    XOR2 = xor(xor(XOR1,D1),D3);
    XOR3 = xor(D2,XOR2);
    XOR4 = xor(D2,D3);
    Y0tail(tl) = XOR2;                  %parity bits written
    Y1tail(tl) = XOR3;
    D3 = D2;                            %shift of registers content
    D2 = D1;
    D1 = XOR1;
end
```

3.3 Turbo interleaver

Pseudo random interleaver is used to permute the data encoded by the second RSC encoder. Block of size $L_{\text{TC-input}}$ input data bits is written to the array and than the entire data sequence is read out in the different order.

Computation of interleaving addresses is presented in the Figure 1c).

Computation of interleaving addresses for block of $L_{\text{TC-input}}$ input data payload bits is obtained in 9 steps. Once the addresses are computed sequence can be saved as *.mat file and used for next runs of FEC encoding initialized by function `load '*.mat'`.

Following Matlab codes perform computation of interleaving addresses for block of $L_{\text{TC-input}}$ input data payload bits according to 9 steps described in chapter 5.3.3 of [1]:

1) Determine the turbo interleaver parameter n, where n is the smallest integer and $L_{\text{TC-input}} \leq 2^{n+5}$.

```
n = 9;                                  %set n according to LTCinp
```

2) Initialize an $(n + 5)$ - bit counter to 0. Presented algorithm differs starting this step. All the needed $(n + 5)$ - bit binary addresses are obtained in advance and following steps are done on the entire block of addresses.

```
INadr = 0:(2^(n+5)-1);                  %initialization of decadic addresses
INbinAll = dec2bin(INadr, (n+5));       %decadic to binary conversion for next step
```

3) Extract the n most significant bits (MSBs) from the counter and add one to form a new value. Then, discard all except the n least significant bits (LSBs) of this value.

```
nMSBs = INbinAll(:,1:n);                %extract n MSB bits
nMSBdec = bin2dec(nMSBs);               %binary to decadic conversion
nMSBdec1 = nMSBdec+1;                   %add one to form a new value
nMSBbin1 = dec2bin(nMSBdec1);           %decadic to binary conversion
[~,b] = size(nMSBbin1);                 %obtaining the number of bits in word
nMSBbin1 = nMSBbin1(:,(b-n+1):end);     %obtaining the n least significant bits
```

4) Obtain the n-bit output of the table look up defined in table with read address equal to the five LSBs of the counter.

```
look_up = [13 335 87 15 15 1 333 11 13 1 121 155 1 175 421 5      %look up table
           509 215 47 425 295 229 427 83 409 387 193 57 501 313 489 391];

LSBs5 = INbinAll(:,(n+1):end);          %5 LSBs bits
LSBdec = bin2dec(LSBs5);                %binary to decadic conversion

for indx = 1:(2^(n+5))
    look_out(indx,1) = look_up((LSBdec(indx)+1));%n bit output of the look up table
end
```

5) Multiply the values obtained in Steps 3 and 4, and discard all except the n LSBs.

```
nMSB1dec = bin2dec(nMSBbin1);           %binary to decadic conversion
multiply = nMSB1dec .* look_out;        %1by1 multiplication
muliplyBin = dec2bin(multiply);         %decadic to binary conversion
[~,b] = size(muliplyBin);
LSBsn = muliplyBin(:,(b-n+1):end);      %obtaining the n least significant bits
```

6) Bit-reverse the five LSBs of the counter.

```
reverse = [5 4 3 2 1];
LSBs5reverse = LSBs5(:,reverse);
```

7) Form a tentative output address that has its MSBs equal to the value obtained in Step 6 and its LSBs equal to the value obtained in Step 5.

```
tentAdr = [LSBs5reverse,LSBsn];         %forming the tentative addresses
tentOutAdr = bin2dec(tentAdr);          %binary to decadic conversion
```

8) Accept the tentative output address as an output address if it is less than LTC-input; otherwise, discard it.

```
indxx = 1;
for indx = 1:(2^(n+5))
    if tentOutAdr(indx) < LTCinp        %accept addresses < LTCinp
    IntOutAdr(indxx) = tentOutAdr(indx);    %interleaver output addresses
    indxx = indxx + 1;                      %addresses in the range 0:12281
    end
end
```

9) Increment the counter and repeat Steps 3 through 8 until all LTC-input interleaver output addresses are obtained. This step is rejected due to different approach to computation algorithm.

3.4 Puncturing patterns

Puncturing patterns *Punct_Pat_ID* 0..11 are defined to limit the redundancy of transmitted data. Systematic bits X and X' contain the same information only with different bits order. Therefore, it is not necessary to transmit booth and only X sequence is transmitted in all Puncturing patterns. Puncturing patterns are defined with the different puncturing period (1 to 12 input data bits) and are repeated with respect to $L_{\text{TC-input}}$. Puncturing and symbol repetition patterns for the tail bit periods are also defined.

Following Matlab codes perform puncturing of Turbo code encoded X, Y_0, Y_1, X', Y'_0 and Y'_1 output data payload bits. Standard pattern *Punct_Pat_ID_6* with CR = 2/5 is presented according chapter 5.3.1 of [1]:

```
Turbo_Encoded_Sequence = [X;Y0;Y1;Xp;Y0p;Y1p];
input_sequence = reshape(Turbo_Encoded_Sequence,1,[]);     %reshape to vector

%puncturing pattern Punct_Pat_ID_6, CR = 2/5
Punct_Pat =[1 0 0 0 0 0; 1 0 1 0 0 1; 0 0 1 0 0 1; 1 0 1 0 0 1;
            1 0 1 0 0 1; 0 0 1 0 0 1; 1 0 1 0 0 1; 1 0 1 0 0 1;
            0 0 1 0 0 1; 1 0 1 0 0 1; 1 0 1 0 0 1; 0 0 1 0 0 1];

[rows,~] = size(Punct_Pat);             %length of pattern

Punc_Pat_Rep = ceil(LTCinp/rows);       %number of patterns repetitions
Punc_Pat_Block = [];                    %initialization of pattern block

for rep = 1:Punc_Pat_Rep
    Punc_Pat_Block = [Punc_Pat_Block; Punct_Pat];    %fill of block by patterns
end

Punc_Pat_Block = Punc_Pat_Block(1:LTCinp,:);     %block shorten to effective length
Punc_Pat_Vector = reshape(Punc_Pat_Block',1,[]); %reshape to vector
Punctured_Out = [];                     %initialization of punctured vector

for indx = 1:length(Punc_Pat_Vector)
    if Punc_Pat_Vector(indx) == 1       %selection of unpunctured bits

        Punctured = input_sequence(indx);
        Punctured_Out = [Punctured_Out, Punctured];  %unpunctured bits written
    end
end
```

4 Turbo decoding process

The DVB-SH Turbo encoding process, based on 3GPP2 is well specified. On the other hand, Turbo code decoding process is not described. Decoding systems are based on Soft-Input Soft-Output (SISO) Decoding and Log Likehood Ratios (LLR). Viterbi algorithm, Maximum a-Posteriori algorithm (MAP), Logarithmic-MAP and other algorithms are used. Information about decoding of Turbo codes can be found in [6], [7] and others.

5 Conclusion

Possible Matlab implementation of DVB-SH Forward Error Correction encoding using Turbo Code is proposed in the paper. It is obvious that other approaches to implementation, different computational algorithm and Matlab built in functions can be used to reach the specified FEC according DVB-SH standard. Turbo code decoding is not presented in the paper as is not standardized and the algorithms used for decoding are considerable complex.

This work should continue by investigation on implementation of the Turbo code decoder and building of Matlab model of the entire DVB-SH Transmission and Receiving system.

Acknowledgement

This paper was supported by the Research programme of the Brno University of Technology no. MSM0021630513, "*Electronic Communication Systems and New Generation Technology (ELKOM)*" and grant project of the Czech Science Foundation no. 102/08/P295, "*Analysis and Simulation of the Transmission Distortions of the Digital Television DVB-T/H*".

References

[1] ETSI EN 302 583 V1.1.2 (2010-02). *Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for Satellite Services to Handheld devices (SH) below 3 GHz.* ETSI, 02/2010. [Online] Available: http://dvb.org/technology/standards/

[2] ETSI EN 300 744 V1.6.1 (2009-01). *Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for digital terrestrial television.* ETSI, 01/2009. [Online] Available: http://dvb.org/technology/standards/

[3] ETSI EN 302 304 V1.1.1. (2004-11). *Digital Video Broadcasting (DVB); Transmission System for Handheld Terminals (DVB-H).* ETSI, 11/2004. [Online]. Available: http://dvb.org/technology/standards/

[4] ETSI EN 302 307 V1.2.1. (2009-08). *Digital Video Broadcasting (DVB); Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications (DVB-S2).* ETSI, 08/2009. [Online]. Available: http://dvb.org/technology/standards/

[5] 3GPP2 C.S0002-D Version 2.0. (2005-09). *Physical Layer Standard for cdma2000 Spread Spectrum Systems. 3GPP2.* 09/2005 [Online]. Available: http://www.3gpp2.org/Public_html/specs/cref.cfm

[6] Valenti, M. C. *Iterative detection and decoding for wireless communications.* Ph.D. Thesis, Bradley Dept. of Elect. & Comp. Eng., Virginia Tech, 07/1999 [Online]. Available: http://www.csee.wvu.edu/~mvalenti/publications/

[7] Loo, K. K., Alukaidey, T., Jimaa, S.A. *High performance parallelised 3GPP turbo decoder.* In proceedings of the 5th European Personal Mobile Communications Conference 2003. p. 337 – 342. ISBN 0-85296-753-5. ISSN 0537-9989. [Online]. Available: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1350211

Contact

Ondřej Hüttl, Tomáš Kratochvíl
Department of Radio Electronics, Brno Univ. of Technology, Purkyňova 118, 612 00 BRNO
E-mail: xhuttl00@stud.feec.vutbr.cz, kratot@feec.vutbr.cz
Tel: +420 541 149 113, Fax: +420 541 149 244