DVB-T Seminar
Topic 7: Frame Building

Carsten Presser

February 13, 2006

Contents

I OFDM Frames 3

1 General information 3

2 Basics for frame building 3

2.1 Scattered Pilot Cells 3
2.2 Continual Pilots 3
2.3 PRBS used in frame building 3
2.4 Pilot modulation 3
2.5 Differential Binary Phase Shift Keying 4
2.6 TPS modulation 4

II Implementation 5

1 Encoder part: dvbt_7_frame.m 5

1.1 Interfaces 5
1.2 Program parts 5
1.2.1 Input buffer 5
1.2.2 Fixed data 6
1.2.3 PRBS Implementation 6
1.2.4 Superfame building: Add pilot and TPS carrier 7
1.3 Problems 9

2 Decoder part: dvbt_7_iframe.m 10

2.1 Interfaces 10
2.2 Program parts 10
2.2.1 Buffers 10
2.2.2 Fixed data 11
2.2.3 TPS demodulation 12
2.2.4 Frame start detection 13
2.2.5 Strip pilot and TPS carriers 13
2.3 Problems 15

III Testing 16

1 General considerations 16

2 Testing encoder and decoder 16

2.1 Implementation 16
2.1.1 Test data 16
2.1.2 Encoding 16
2.1.3 Decoding 16
2.2 Problems 17

Part I
OFDM Frames

1 General information

Data in a DVB-T transmission ist organized in frames and superframes.
Each frame consits of 68 Symbols, numbered from 0..67. A superframe contains 4 of these frames.
In addition to the transmitted data, an ODFM frame includes TPS carriers, scatterd pilot cells and continual pilot carriers.

In 2K mode, there are 1705 active carriers; in 8k mode there are 6817 active carriers. Out of these 1705 (6817) carriers, only 1512 (6048) contain payload.

2 Basics for frame building

2.1 Scattered Pilot Cells

For the symbol of index l (ranging from 0 to 67), carriers for which index k belongs to the subset

$$k = K_{min} + 3 \cdot (l \mod 4) + 12p, p \in \mathbb{N}, p \geq 0, k \in [K_{min}; K_{max}] \tag{1}$$

are scattered pilots. Where p is an integer that takes all possible values greater than or equal to zero.

2.2 Continual Pilots

In addition to the scattered pilots 45 (2k) or 177 (8k) pilots are inserted as shown in table XXX. These pilots are on the same position in *every* symbol.

2.3 PRBS used in frame building

The PRBS is used to modulate both pilot and TPS carriers. The PRBS is initialized so that the first output bit from the PRBS coincides with the first active carrier. A new value w_k is generated by the PRBS on *every* carrier (payload, pilot or TPS).

2.4 Pilot modulation

Pilots are transmitted at boosted power level. The modulation is given by:

$$\Re = \frac{4}{3} \cdot 2\,(0.5 - w_k) \tag{2}$$

$$\Im = 0 \tag{3}$$

w_k is the output bit provided by the PRBS.

2.5 Differential Binary Phase Shift Keying

DBPSK is a modulation with one bit per symbol. The modulation scheme itself is very easy: a logical one is represented by a change (shift) of the phase, a logical zero is represented by the same phase. So the actual bit is not encoded on a symbol, but between two symbols (a change can only occur between two symbols). Because of this fact, the first symbol (initialization) in a DBPSK modulation does not convey any information. This is the same in decoding, the first symbol can't be decoded; the first bit arrives with the second symbol.

Demodulating is also a simple task: just check if the phase shifted between two symbols.

2.6 TPS modulation

Unlike the pilot carriers, TPS carriers are transmitted at a normal power level. In a Symbol each TPS carrier conveys the same information. The TPS data itself is DBPSK modulated.

In addition to the great redundancy achieved through distributing the TPS information over multiple carriers, the TPS word itself is protected with a BCH-code, which will be processed in module 6 *dvbt_6_itps.m*.

Each of the 68 TPS bits is modulated in one of the 68 symbols in a frame. The modulation scheme for the first symbol (initialization) is the same as the pilot carrier modulation (expect the power-level):

$$\Re = 2\,(0.5 - w_k) \tag{4}$$

$$\Im = 0 \tag{5}$$

The TPS carriers in the following symbols ($l > 0$) are modulated with the following rule:

$$\Im_l = 0 \tag{6}$$

$$\Re_{(l)} = \begin{cases} \Re_{(l-1)} & s_l = 0, \\ -1 \cdot \Re_{(l-1)} & s_l = 1 \end{cases} \tag{7}$$

s_l is the TPS bit for the symbol l.

Part II
Implementation

1 Encoder part: dvbt_7_frame.m

The function *dvbt_7_frame.m* is the 7th function in the encoder pipeline. It's task is to add TPS information (provided by module 6) and pilot (static) to the symbols generated by module 5. As the TPS information is cyclic with every superframe, the program collects input data from module 5 until enough symbols to build a complete superframe are in its input buffers.

1.1 Interfaces

The input data needs to be one symbol: a 1x1512 (2k-mode) or 1x6048 (8-k-mode) matrix. Once the program has collected enough input data (272 symbols), it starts to generate a superframe.
A superframe is a 272x1705 (2k-mode) or 272x6817 (8k-mode) matrix with complex entrys.

1.2 Program parts

1.2.1 Input buffer

The Encoder has to persistent variables: *sf_raw* and *sf_raw_cnt*.

```
persistent sf_raw;                        % these are my buffers
persistent sf_raw_cnt;

```

In Case these Variables are empty, they are initialized.

```
% init buffers in case they are empty
if isempty(sf_raw_cnt)
  sf_raw = zeros(nSymbolsSF,wi);
  sf_raw_cnt = 0;
end

```

Input data will be collected until the buffer is filled and we can start to build a complete superframe. After that, the input buffer is reset, so we can start collecting data for the next superframe. In case the buffer is not filled up, the programm exits to request further input.

```
% now collect data and fill up my personal input buffer
if sf_raw_cnt < nSymbolsSF
  sf_raw_cnt = sf_raw_cnt + 1;
  sf_raw(sf_raw_cnt,:) = sym;
  debug(sprintf ('[dvbt_7_frame] inputbuffer @%2.1f %%\n',(sf_raw_cnt/nSymbolsSF*100)),
  if sf_raw_cnt < nSymbolsSF
    sf = [];
    return;
  end
end
sf_raw_cnt = 0;

```

1.2.2 Fixed data

In Order to insert TPS and pilot information we need the carrier numbers as described in ??.

```
% these are the carrier numbers taken from the spec (i HATE typing stupid numbers)
TPS_FIXED_2K = [34,50,209,346,413,569,595,688,790,901,1073,1219,1262,1286,1469,1594,168
TPS_FIXED_8K = [34,50,209,346,413,569,595,688,790,901,1073,1219,1262,1286,1469,1594,168
PILOT_FIXED_2K = [0,48,54,87,141,156,192,201,255,279,282,333,432,450,483,525,531,618,63
PILOT_FIXED_8K = [0,48,54,87,141,156,192,201,255,279,282,333,432,450,483,525,531,618,63
% switch mode
if DVB_SETTINGS.mode == '2'
  PILOT_FIXED = PILOT_FIXED_2K + 1;
  TPS_FIXED = TPS_FIXED_2K + 1;
  TPS_FIXED_CNT = 20;
  PILOT_FIXED_CNT = 45;
elseif DVB_SETTINGS.mode == '8'
  PILOT_FIXED = PILOT_FIXED_8K + 1;
  TPS_FIXED = TPS_FIXED_8K + 1;
  TPS_FIXED_CNT = 68;
  PILOT_FIXED_CNT = 177;
end

```

These variables have the following meaning:

TPS_FIXED_2K The carrier numbers for fixed TPS carriers in 2k-mode

TPS_FIXED_8K The carrier numbers for fixed TPS carriers in 8k-mode

PILOT_FIXED_2K The carrier numbers for fixed pilot carriers in 2k-mode

PILOT_FIXED_8K The carrier numbers for fixed pilot carriers in 8k-mode

TPS_FIXED_CNT The absolute number of fixed TPS carriers

PILOT_FIXED_CNT The absolute number of fixed pilot carriers

TPS_FIXED The carrier numbers for fixed TPS carriers in the selected mode (Matlab start to count array indices with 1)

PILOT_FIXED The carrier numbers for fixed pilot carriers in the selected mode (Matlab start to count array indices with 1)

1.2.3 PRBS Implementation

The PRBS ist initialzied with every 11 bits as '1'. The PRBS istself is stored as a simple integer in the varbiable *PRBS_register*.

```
 90   % start the pseudo-random-number-generator (see 4.5.2)
 91   debug(sprintf('[dvbt_7_frame] initializing the pseudo-random-number-generator...\n'),1)
 92   PRBS_register = 2^12 - 1;
 93
```

A New bit is generated on line 123:

```
123   wk = bitget(PRBS_register,11:1:11);
124   PRBS_register = bitshift(PRBS_register,1) + xor(bitget(PRBS_register,11:1:11),bitget(PR
125
```

Here simply the last bit of the stored value is taken. After that, we shift the whole register and add a new first bit with the xor operation on bit 11 and 9. As we are already inside the *CarrCnt* loop, wich iterates over all carriers, a new bit is generated on every carrier, as demanded in 2.3.

1.2.4 Superfame building: Add pilot and TPS carrier

The main part of the program consits of 3 nested loops:

FrameCnt This loop runs 4 times. In every cycle, one frame is processed, producing a complete superframe.
The only task in this loop part is to get the TPS word, as it changes for every frame embedded in a superframe.

SymbolCnt The SymbolCnt loop iterates over the 68 symbols of a frame. Each time this loop starts, we reset some counter used in the inner *CarrCnt* loop. The most important part here is to calculate the number of the first scattered pilot in the current symbol, as done in line 113:

```
113   % for the scattered pilots: get the index of the FIRST scattered pilot:
114   % HINT: we need to add one, because matlab (and i) start to count carriers by 1.
115   %       the same goes for (SymbolCnt - 1)
116   sc_pilot_1 = 1 + 3 * mod((SymbolCnt-1),4);
117
```

CarrCnt The innermost loop runs over the numer of active carriers, respective to the selected mode. The Algorithm is very simple: 3 checks are done, if the current carrier is either fixed TPS, scatterd pilot or fixed pilot. If not, the modulated payload is inserted:

```
170   %
171   % okay... now we reached here means: this symbol actually contains payload.
172   %
173   carrier_cnt = carrier_cnt + 1;
174   sf(CurrSymbolNum,CarrCnt) = sf_raw(CurrSymbolNum,carrier_cnt);
175
```

As mentioned above, there are 3 checks bevore this:

fixed pilot A simple check is done, if the current carrier matches one of the the fixed pilots which are stored in *PILOT_FIXED*. If yes, the carrier is modulated and we continue to process the next active carrier.

```
%
% check for fixed pilot
%
if CarrCnt == (PILOT_FIXED(pilot_cnt+1))
    % looks like this carrier is a FIXED pilot carrier
    pilot_cnt = pilot_cnt + 1;
    sf(CurrSymbolNum,CarrCnt) = pilot_data_mod;
    continue;
end

```

scatterd pilot This is the same as for fixed pilots, expect the check if the current carrier is a scatterd pilot. Because scattered pilots occur on every 12th carrier, this check can be easyly done with a modulo division:

```
%
% check for scattered pilot
%
if mod((CarrCnt - sc_pilot_1),12) == 0
    % looks like this carrier is a SCATTERED pilot carrier
    sf(CurrSymbolNum,CarrCnt) = pilot_data_mod;
    continue;
end

```

fixed TPS Checking if the current carrier is a TPS carrier works analog to the fixed pilot carriers. The only change is the modulation (pilot carriers don't carry any usefull data). Because the TPS carriers are DBPSK modulated, a different code path is needed for the first and the rest of the symbols.

```
%
% check for TPS
%
  if CarrCnt == (TPS_FIXED(tps_cnt+1))
    % in case we are here for the first time, we need to modulate the TPS carr
    if SymbolCnt == 1
      % modulate TPS-Bit S0
      tps_data_mod(CarrCnt) = 2 * (1/2 - wk);
    else
      % do a DBPSK
      if tps_word(SymbolCnt) == 1
      tps_data_mod(CarrCnt) = -1 * tps_data_mod(CarrCnt);     % shift phase
    end
  end
  % looks like this carrier is a FIXED tps-carrier
  tps_cnt = tps_cnt + 1;
  sf(CurrSymbolNum,CarrCnt) = tps_data_mod(CarrCnt);
  continue;
end

```

1.3 Problems

In case the mode (2k or 8k) is switched and the modules input buffers are not cleared manually, the program will encounter an error because it tryes to add data in a matrix of incorrect size. Because this is not a killer bug, this issue has not been solved. Some other modules seem to have the same problem to. This can be workarounded by executing a 'clear all' right before a encoder run.

2 Decoder part: dvbt_7_iframe.m

The decoder program dvbt_7_iframe.m does the exact opposite of the encoder dvbt_7_frame.m. It removes pilot information and decodes TPS data.

2.1 Interfaces

The input data needs to be one symbol: a 1x1705 (2k-mode) or 1x6817 (8-k-mode) matrix. Once the program has collected a complete frame, it starts the actual decoding. A frame is a 68x1512 (2k-mode) or 68x6048 (8k-mode) matrix with complex entrys.

2.2 Program parts

2.2.1 Buffers

Because of the FrameDetection Algorithm, i is necessary to store some more persistent data.

```
%
% STEP 0: Get global & persistent data;
%
global DVB_SETTINGS;                   % import data
global DVB_TPS_STRUCT;
persistent f_raw;
persistent f_raw_cnt;
persistent f_raw_cnt_disc;
persistent frame_detected;
persistent tps_data_mod;
persistent tps_data;


```

Buffer initialization and filling is done analog to the encoder:

```
% init buffers in case they are empty
if isempty(f_raw_cnt)
    % we collect input data here
    f_raw = zeros(nSymbols,wi);
    f_raw_cnt = 0;
    f_raw_cnt_disc = 0;

    % init a matrix holding TPS-modulation information (we need this for PSK)
    tps_data_mod = zeros(1,wo);

    % init a matrix holding the DEmodulated TPS-Bits
    tps_data = zeros(nSymbols,1,'int8');
    tps_data(:,1) = -1; % set to undefined

    % a marker: did we already detect a frame?
    frame_detected = 0;
end
```

```

% now collect data and fill up my personal input buffer
f_raw_cnt = f_raw_cnt + 1;
f_raw(f_raw_cnt,:) = sym;
debug(sprintf ('[dvbt_7_iframe] inputbuffer: %d symbols\n',f_raw_cnt),2);


```

The persistent variables have the following purpose:

f_raw This matrix buffers the input data. One line equals one Symbol.

f_raw_cnt is a counter for the number of symbols in the input matrix *f_raw*.

f_raw_cnt_disc a counter for the number of discarded symbols.

frame_detected This marker switches to '1' when a frame start has been detected.

tps_data_mod A Matrix containing the modulated TPS values from the last symbol. This matrix is required because TPS is modulated with DBPSK (I need to know the last state to demodulate).

tps_data This Vector contains the demodulated TPS data.

The buffers are cleared, once a complete frame has been processed.

2.2.2 Fixed data

Same as in the decoder, we need to know the fixed places of TPS and pilot carriers. Also a sync word, to detect the start of a frame is defined.

```
%
% STEP 1: Prepare some fixed data from the spec
%
debug(sprintf('[dvbt_7_iframe] generating fixed pilot&tps information...\n'),1);
% these are the carrier numbers taken from the spec (i HATE typing stupid numbers)
TPS_FIXED_2K = [34,50,209,346,413,569,595,688,790,901,1073,1219,1262,1286,1469,1594,168
TPS_FIXED_8K = [34,50,209,346,413,569,595,688,790,901,1073,1219,1262,1286,1469,1594,168
PILOT_FIXED_2K = [0,48,54,87,141,156,192,201,255,279,282,333,432,450,483,525,531,618,63
PILOT_FIXED_8K = [0,48,54,87,141,156,192,201,255,279,282,333,432,450,483,525,531,618,63

TPS_SYNC_WORD =     [0,0,1,1,0,1,0,1,1,1,1,0,1,1,1,0]';
TPS_NOT_SYNC_WORD = [1,1,0,0,1,0,1,0,0,0,0,1,0,0,0,1]';

% switch mode
if DVB_SETTINGS.mode == '2'
    PILOT_FIXED = PILOT_FIXED_2K + 1;
    TPS_FIXED = TPS_FIXED_2K + 1;
    TPS_FIXED_CNT = 17;
    PILOT_FIXED_CNT = 45;
elseif DVB_SETTINGS.mode == '8'
    PILOT_FIXED = PILOT_FIXED_8K + 1;
    TPS_FIXED = TPS_FIXED_8K + 1;
    TPS_FIXED_CNT = 68;
    PILOT_FIXED_CNT = 177;
end

```

2.2.3 TPS demodulation

TPS demodulation is the most important part in the decoding process. Unless the TPS information is correct, no other module can work properly.

In theory each carrier holding TPS information should convey the same bit. Because of possible noise and disturbance during transmission, it is possible that single bits won't be correct. Therefore, we try to workaround this by using a majority rule. The absolute numbers of logical zero and logical one are counted and compared. The state, which occurred more often wins. Because this is a relative simple rule, debugging information is given, if one of the state does not win with at least 60% of the absolute number of sample.

The actual demodulating is done by calculating the L2 distance between the current sample and the sample collected in the last symbol. If the distance is greater than one (in a ideal transmission, the distance will either be zero or two) the program considers this as a logical one (otherwise: logical zero).

Because TPS is modulated with DBPSK, the first symbol can't contain any information, the demodulation starts with symbol two.

```
if f_raw_cnt == 1
    % we are on the first symbol. we just need to init the tps_data_raw
    % vector. decoding ist NOT possible.
    for CarrCnt = TPS_FIXED
        tps_data_mod(1,CarrCnt) = sym(1,CarrCnt);
    end % CarrCnt
else
    tps_cnt = [0,0];    % we need this for the majority rule
    for CarrCnt = TPS_FIXED
        % simple PSK-demodulation. hope this works
        if abs(real(tps_data_mod(1,CarrCnt)) - real(sym(1,CarrCnt))) > 1
            tps_cnt (1,2) = tps_cnt (1,2) + 1;
        else
            tps_cnt (1,1) = tps_cnt (1,1) + 1;
        end

        % update 'old' tps_data_mod (sounds funny :P)
        tps_data_mod(1,CarrCnt) = sym(1,CarrCnt);
    end % CarrCnt

    % error detection... do a majority rule
    [num,idx] = max (tps_cnt);
    tps_data(f_raw_cnt,1) = idx - 1;

```

2.2.4 Frame start detection

Because the sync word is 16 bits long, we can't decide if a frame has started until the input buffer holds at least $nSync = 17$ [1] symbols.

```
if f_raw_cnt < nSync
    % we cant decide on a framestart... yet
    frame = [];
    return;
end

```

In case no start can be detected the first symbol in the input buffer is discarded (FIFO). The other symbols in the input buffer are shifted one position down. The next symbol arriving in the decoder will again be symbol number 17.

```
f_raw_cnt_disc = f_raw_cnt_disc + 1;
debug(sprintf('[dvbt_7_iframe] no frame start yet (already discarded %d symbols)...\n',
%shift input matrix, discard the 'old' symbol number one
f_raw (1:(nSync-1),:) = f_raw(2:nSync,:);   % (i know, this looks ugly)
f_raw (nSync,:) = zeros (1,wi);
f_raw_cnt = nSync -1;

% shift already decoded TPS-data
tps_data(1:(nSync-1),1) = tps_data(2:nSync,1);
tps_data(nSync) = -1;

```

Detecting a frame start is a simple compare operation of the decoded TPS data and the sync word. In case they are the same, a frame start has been found.

```
% check for frame-start...
diff = [tps_data(2:nSync,1),TPS_SYNC_WORD,TPS_NOT_SYNC_WORD];
if diff(:,1) == diff(:,2)
    debug(sprintf('[dvbt_7_iframe] start of a Frame detectet! (1,3)\n'),4);
    frame_detected = 1;
elseif diff(:,1) == diff(:,3)
    debug(sprintf('[dvbt_7_iframe] start of a Frame detectet! (2,4)\n'),4);
    frame_detected = 1;

```

2.2.5 Strip pilot and TPS carriers

This is just the inverse process, as described in the encoder documentation under 1.2.4. One difference is, that there are only two loops. This is because in *dvbt_7_iframe.m* only one frame is processed (not a superframe). Same as in 1.2.4, there are three checks (fixed pilots, fixed TPS, scattered pilots). If the carrier is not out of one of these groups, it is considered as payload and added to the output matrix.

[1]NOT the BoyBand. Metal rul0rz!

```
for SymbolCnt = 1:1:68
    % reset some local counters
    tps_cnt = 0;
    pilot_cnt = 0;
    carrier_cnt = 0;

    % for the scattered pilots: get the index of the FIRST scattered pilot:
    % HINT: we need to add one, because matlab (and i) start to count carriers by 1.
    %       the same goes for (SymbolCnt - 1)
    sc_pilot_1 = 1 + 3 * mod((SymbolCnt-1),4);


    %
    % strip the symbol...
    %
    for CarrCnt = 1:1:wi
        %
        % check for TPS
        %
        if CarrCnt == (TPS_FIXED(tps_cnt+1))
            tps_cnt = tps_cnt + 1;
            continue;
        end

        %
        % check for fixed pilot
        %
        if CarrCnt == (PILOT_FIXED(pilot_cnt+1))
            pilot_cnt = pilot_cnt + 1;
            continue;
        end

        %
        % check for scattered pilot
        %
        if mod((CarrCnt - sc_pilot_1),12) == 0
            continue;
        end

        %
        % okay... now we reached here means: this carrier actually contains payload.
        %
        carrier_cnt = carrier_cnt + 1;
        frame(SymbolCnt,carrier_cnt) = f_raw(SymbolCnt,CarrCnt);  % frame DOES now cont

        % done. this carrier should now be stripped.

    end  % CarrCnt
    debug(sprintf('[dvbt_7_frame] %d payload carriers processed (should be 1512 or 6048
end % SymbolCnt
```

2.3 Problems

Here the same error as in the encoder (1.3) occurs. A 'clear all' before the decoder run may help.

Part III
Testing

1 General considerations

Because every module in the pipeline uses its own interfaces, a generalized testing routine is not possible. Also most modules need a different amount of data to start working.

These two things make testing the whole pipeline difficult, therefore a single test for every module has to be created.

2 Testing encoder and decoder

An easy way to test both, encoder and decoder, would be to encode some test data, decode them and compare the decoded data with the original test data. This test has to be repeated for every global setting, that influences the module. In case of module 7, this is only the mode setting (2k- or 8k-mode).

2.1 Implementation

2.1.1 Test data

The test data is generated as four sets of continual numbers. Two of these sets are changed into complex numbers by adding I.

```
% generate some test-symbols
tsym_1 = 0.001:0.001:(wi*1)/1000;
tsym_2 = (1+wi*1)/1000:0.001:(wi*2)/1000;
tsym_3 = (1+wi*2)/1000:0.001:(wi*3)/1000;
tsym_4 = (1+wi*3)/1000:0.001:(wi*4)/1000;

% make symbol 1 and 3 complex
tsym_1 = tsym_1 + j;
tsym_3 = tsym_3 + j;
```

2.1.2 Encoding

The encoder is loaded with a repeating cycle of these four test symbols. Once its input buffers are filled, it should start decoding[2].

2.1.3 Decoding

A loop is begin run over the matrix generated by the encoder. By changing the start index of this loop, testing of the frame detection is possible. In case the loop starts with index one, the decoder should find a frame start after 17 symbols, also not a single symbol should get discarded. Another possibility would be to start at index 67: the first symbol still belongs to the first frame,

[2]this is already part of the test

so it should be discarded. The frame start should be detected after symbol 18 (which is symbol 17 of the 2nd frame).

2.2 Problems

The test routine needs user interaction.

First of all, the mode will not be switched automatically. To run a complete test, the mode has to be changed in the file *dvbt_config_encoder.m* manually

Also, in case not everything goes the way it is supposed to to, a closer look at the output of the script is required. The TPS check has not been automated to far, the user has to ensure if one of the TPS-lines in the output equals all zero.