ISO/IEC JTC 1/SC 17
Cards and personal identification
Secretariat: BSI (United Kingdom)

Document type: Text for PDAM ballot or comment

Title: Notification of ballot: ISO/IEC 10373-6:2011/PDAM5.2 — Identification cards — Test methods — Part 6: Proximity cards — AMENDMENT 5: Bit rates of 3fc/4, fc, 3fc/2 and 2fc from PCD to PICC

Status:

BACKWARD POINTER: N 4253, N 4312 and N 4575.

STATUS: This ballot has been posted to the ISO Electronic balloting application and is available under the Balloting Portal, Committee Internal Balloting.

WORK ITEM: 60115

Date of document: 2012-06-20

Expected action: VOTE

Action due date: 2012-09-21

No. of pages: 31

Email of secretary: chris.starr@ukpayments.org.uk

Committee URL: http://isotc.iso.org/livelink/livelink/open/jtc1sc17

ISO/IEC JTC 1/SC 17/WG 8 N1899

Date: 2011-06-08

ISO/IEC 10373-6:2011/PDAM 5.2

ISO/IEC JTC 1/SC 17/WG 8

Secretariat: DIN

Identification cards — Test methods — Part 6: Proximity cards

AMENDMENT 5
Bit rates of 3*fc*/4, *fc,* 3*fc*/2 and 2*fc* from PCD to PICC

Cartes d'identification — Méthodes d'essai — Partie 6: Cartes de proximité

AMENDEMENT 5
Débits binaires de 3fc/4, fc, 3fc/2 et 2fc de PCD vers PICC

Document type: International Standard
Document subtype: Amendment
Document stage: (30) Committee
Document language: E

C:\Userdata\Standards\10373-6\Amendment 5 VHBR PSK\Editing Graz\Provided by NXP\wg8n1899_CD2_10373-6_PDAM5_2_VHBR_PSK.doc STD Version 2.1c2

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Amendment 5 to ISO/IEC 10373-6:2011 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 17, *Cards and personal identification*.

Identification cards — Test methods — Part 6: Proximity cards

Page 22

Add new subclause

"

7.3 Test methods for bit rates of 3*fc*/4, *fc*, 3*fc*/2 and 2*fc* from PCD to PICC

See Annex J.

"

After Annex I

Add following new annex:

Annex J
(normative)

Test methods for bit rates of 3*fc*/4, *fc*, 3*fc*/2 and 2*fc* from PCD to PICC

J.1 Overview

This annex specifies the test methods for bit rates of 3*fc*/4, *fc*, 3*fc*/2 and 2*fc* from PCD to PICC.

J.2 Test of ISO/IEC 14443-2 parameters

J.2.1 PCD Tests

All the tests described below will be done in the operating volume as defined by the PCD manufacturer.

J.2.1.1 PCD phase range and waveform characteristics

J.2.1.1.1 Purpose

This test is used to determine the PR transmitted by the PCD as well as the normalized differential phase noise and inter-symbol interference parameters, ISI_m and ISI_d, as defined in ISO/IEC 14443-2:2010/Amd 5.

J.2.1.1.2 Test procedure

Apply the procedure defined in 7.1.4.2 with the following adaptations:

— After the activation of a bit rate of 3*fc*/4, *fc*, 3*fc*/2 or 2*fc*, the PCD shall transmit I(0)0(TEST_COMMAND1(1)).

— In steps a) and f) of 7.1.4.2, the waveform characteristics shall be determined using the analysis tool defined in J.3.

J.2.1.1.3 Test report

The test report shall give the measured PR, ISI_m, ISI_d and the normalized differential phase noise values of the PCD field, within the defined operating volume in unloaded and loaded conditions. The measured parameters shall be within the limits for the PCD as specified in ISO/IEC 14443-2:2010/Amd 5.

NOTE J.3.12 gives some example test reports.

J.2.2 PICC Tests

J.2.2.1 PICC reception

J.2.2.2 Purpose

The purpose of this test is to verify the ability of the PICC to receive PCD commands for bit rates of 3*fc*/4, *fc*, 3*fc*/2 and 2*fc*.

J.2.2.3 Test conditions

Five test conditions are defined at the border of the PICC signal parameters as defined in ISO/IEC 14443-2:2010/Amd 5. The test conditions are created using the test PCD assembly for bit rates higher than *fc*/128 in combination with digital pre-conditioning of the transmitted NPVs as shown in J.4.

— Condition 1: A low pass filtered pseudo-random white noise as defined in J.4.3 is added to the transmitted NPVs such that the normalized differential phase noise (rms) is the maximum value as defined in ISO/IEC 14443-2:2010/Amd 5.

— Condition 2: Maximum ISI_m for $0° \leq ISI_d \leq 90°$: The test PCD signal is digitally pre-conditioned to have the maximum ISI_m value for the ISI_d value of 45° as defined in ISO/IEC 14443-2:2010/Amd 5.

— Condition 3: Maximum ISI_m for $-90° \leq ISI_d \leq 0°$: The test PCD signal is digitally pre-conditioned to have the maximum ISI_m value for the ISI_d value of -45° as defined in ISO/IEC 14443-2:2010/Amd 5.

— Condition 4: Maximum ISIm for ISId > 90°: The test PCD signal is digitally pre-conditioned to have the maximum ISI_m value for the ISI_d value of 120° as defined in ISO/IEC 14443-2:2010/Amd 5.

— Condition 5: Maximum ISI_m for $ISI_d = 0°$: The test PCD signal is digitally pre-conditioned to have the maximum ISI_m value for the ISI_d value of 0° as defined in ISO/IEC 14443-2:2010/Amd 5.

NOTE These conditions are applied after switching to the bit rate under test.

NOTE J.4 informatively describes how to create the above 5 conditions in the base-band domain (on the complex envelope of the signal).

These 5 test conditions shall be tested at least using H_{min} and H_{max}.

J.2.2.4 Test procedure

A PICC supporting the optional bit rates of 3*fc*/4, *fc*, 3*fc*/2 and 2*fc* shall operate under the defined conditions after the selection of a respective bit rate. This PICC shall respond correctly to an I(0)0(TEST_COMMAND1(1)) transmitted at the specified bit rate.

The activation of the bit rates uses S(PARAMETERS) mechanism as defined in ISO/IEC 14443-4.

NOTE For testing, the test PCD shall transmit an I(0)0(TEST_COMMAND1(1)). For a frame size higher than 256 Bytes preferable a frame with error correction as defined in ISO/IEC 14443-4 should be used.

J.2.2.5 Test report

The test report shall confirm the intended operation at the bit rates under test. Used test conditions shall be mentioned in the test report.

J.3 Analysis tool for bit rates of 3*fc*/4, *fc*, 3*fc*/2 and 2*fc*

J.3.1 Overview

The working principle of the analysis tool for bit rates of 3*fc*/4, *fc*, 3*fc*/2 and 2*fc* is illustrated in Figure J.1.

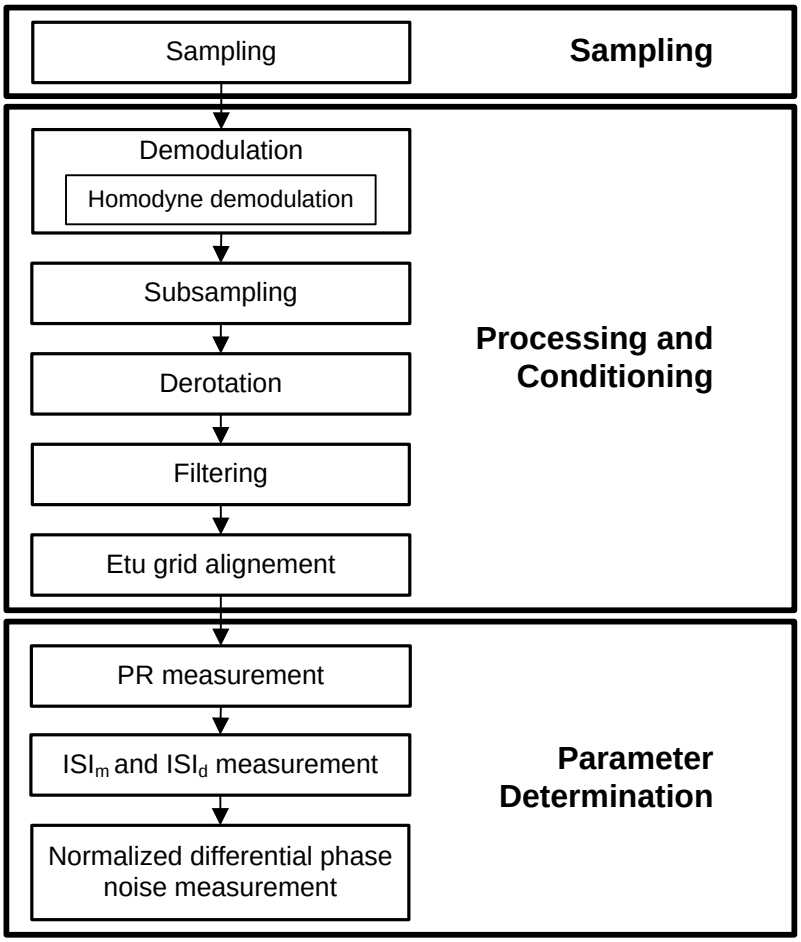

Figure J.1 — Block diagram of the analysis tool for bit rates of 3*fc*/4, *fc*, 3*fc*/2 and 2*fc*

Each block is separately described in the subsequent clauses.

J.3.2 Sampling

The oscilloscope used for signal capturing shall fulfill the requirements defined in 5.1.1. The time and voltage data of 1000 non-modulated carrier periods followed by one data frame, followed by 10 non-modulated carrier periods (see illustration in Figure J.2) shall be transferred to a suitable computer.

Figure J.2 — Non-modulated carrier followed by one frame, followed by non-modulated carrier

J.3.3 Demodulation

The signal shall be demodulated using a homodyne demodulator (IQ demodulator) and the argument of this complex transform represents the phase signal over time (see Figure J.3).

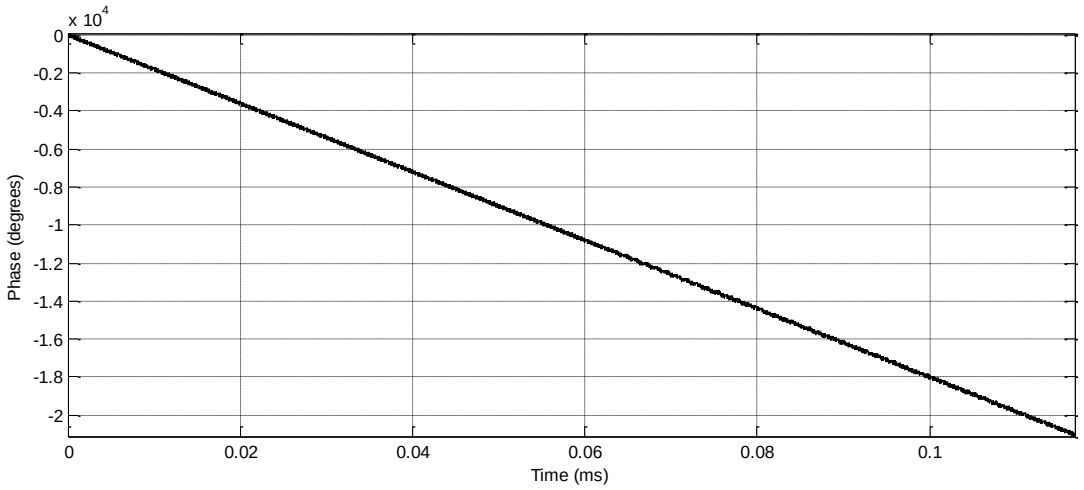

Figure J.3 — Phase signal after homodyne demodulation

J.3.4 Subsampling

The phase signal shall be sub-sampled to an integer number multiple of *fc* using linear interpolation. The integer number shall be equal or greater than 32.

J.3.5 Derotation

This phase signal over time is rotating due to the not synchronized sampling of the oscilloscope. The phase signal shall be derotated to compensate for this phase rotation due to the sampling process (see Figure J.4).

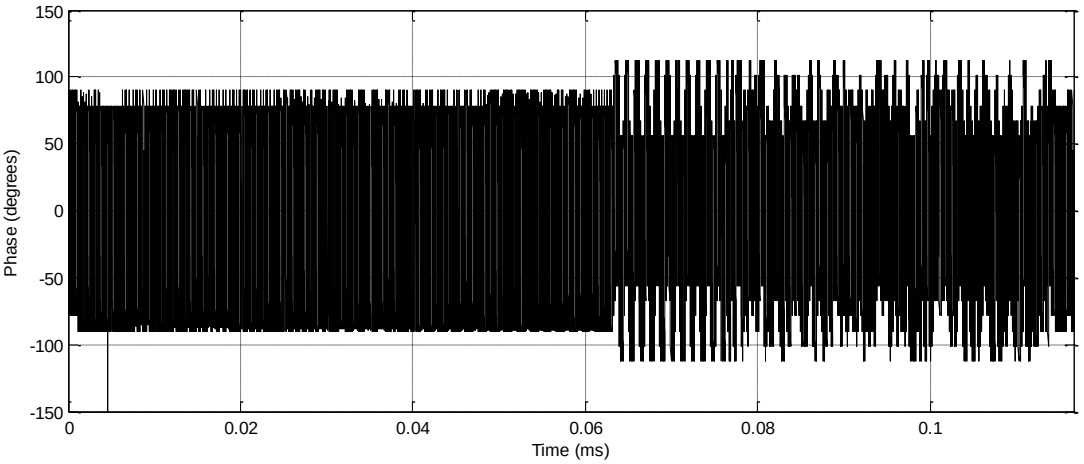

Figure J.4 — Phase signal after Derotation

J.3.6 Filtering

The phase signal contains the 2nd harmonic due to demodulation. A finite impulse response type moving average low-pass filter with cut-off frequency at 12 MHz and a first Null at 2*fc* shall be used for filtering the 2nd

harmonic and higher frequency components. The filter characteristic is illustrated in Figure J.5. Figure J.6 shows the filtered phase signal.

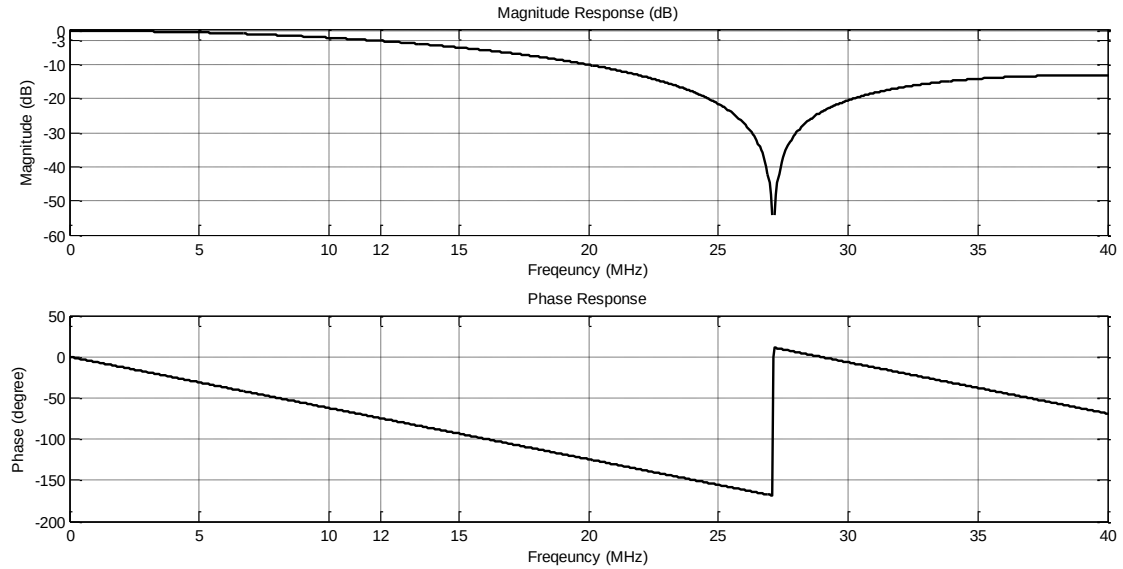

Figure J.5 — Low-pass filter characteristics

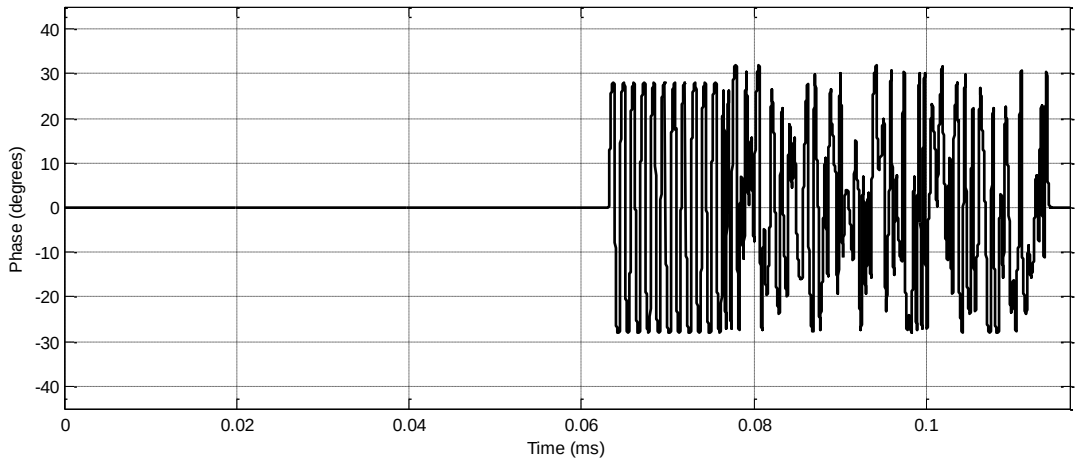

Figure J.6 — Phase signal after Filtering

J.3.7 Etu grid alignment

The phase signal shall be aligned to the etu grid of the reference phase signal. The reference phase signal is computed from the SOC and the data of the frame using the method defined in ISO/IEC 14443-2:2010/Amd 5. The etu grid alignment is carried out by maximizing the correlation cost function computed using the phase signal and the reference phase signal.

J.3.8 PR measurement

The phase range shall be determined as defined in ISO/IEC 14443-2:2010/Amd 5.

J.3.9 ISI_m and ISI_d parameters measurement

The ISI_m and ISI_d parameters shall be determined from the system identification coefficients. The system identification coefficients shall be determined by solving the system identification problem given by the phase signal and the reference phase signal using the Linear Least Squares method. ISI_m and ISI_d values shall be computed for every sampling time within the last carrier period of an etu. The maximum ISI_m value shall be selected with the related ISI_d.

J.3.10 Normalized differential phase noise measurement

The normalized differential phase noise shall be determined during a section of a non-modulated carrier of at least 500 carrier periods according to the definition in ISO/IEC 14443-2:2010/Amd 5.

J.3.11 Program of the analysis tool for bit rates of 3*fc*/4, *fc*, 3*fc*/2 and 2*fc* (informative)

The following program written in MATLAB language gives an example for the implementation of the analysis tool for bit rates of 3*fc*/4, *fc*, 3*fc*/2 and 2*fc*.

This MATLAB implementation consists of 13 files which should be placed in the same folder.

```
function [ISIm, ISId, nd_PN_rms, PR] = mainVHBR_PSK(in, fs, bit_rate)

% function [ISIm, ISId, nd_PN_rms, PR] = mainVHBR_PSK(in, fs, bit_rate)
%
% DESCRIPTION:
%
% INPUT:
%       in...recorded samples from Oscilloscope
%       fs...sampling frequency
%       bit_rate...bit_rate used to modulate 'in';
%       supported bit rates: '3fc/4','fc','3fc/2','2fc'
% OUPUT:
%       ISIm...measured inter symbol interference magnitude value [EPI]
%       ISId...measured inter symbol interference rotation value [degree]
%       nd_PN_rms...measured normalized differential phase error (rms)
%       [EPI]
%       PR...phase range [degree]
%

in = in(:); % make column vector
fc = 13.56e6; % carrier frequency [Hz]

[etu, order, PR_ref, Ph_ref, EPI] = get_bit_rate_param(bit_rate, fc); % get PSK related parameter

% load reference SOC data
[y_ref, phi_ref, SOC_plain] = LUT_SOC_vhbrPsk(order);



alg_type = 'fft'; % select: 'polyfit', 'fft', 'slope'
LP_type = 'moving_avg'; % select: 'moving_avg', 'gaussian', 'LP_massimo', 'fir2_LP'
idx_unmod = round(500*fs/fc); % assume at least 500 unmodulated carrier periods at the beginning

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Processing and Conditioning
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% ------------------------------------------- %
% Homodyne Demodulation
t = [0:length(in)-1]/fs;
[in_bb] = IQ_demodulation(in, t, fc);
in_bb = in_bb.*exp(-1i*2*pi*100e3*t');
% ------------------------------------------- %

% ------------------------------------------- %
% subsampling

% resample to 32 times fc
fs_int = 32*fc;
```

```
fact = fs_int/fs;

% perform sampling to integer sample rate
t_int = [0:ceil(fact*(length(in_bb)-1))]'/fs_int;

in_bb_int = interp1(t, in_bb, t_int, 'linear');
% ---------------------------------------------- %


% ---------------------------------------------- %
% Derotation

[in_bb_int, f_err] = derotation(in_bb_int, idx_unmod, fs_int, fc);

% ---------------------------------------------- %


% ---------------------------------------------- %
% Filtering: perform low-pass filtering to remove 2*fc

[b,a] = LP_filter(LP_type, fs_int, fc); % get low-pass filter coefficients

x = filter(b,a,real(in_bb_int)) + 1i*filter(b,a,imag(in_bb_int)); % filter signal

in_bb_int = x(length(b)+1:end); % remove filter fade-in transient from signal

% perform rotation such that silence carrier has phase 0 degrees
phi_error = mean((angle(in_bb_int(1:idx_unmod))));
in_bb_int = in_bb_int*exp(-1i*phi_error);

% ---------------------------------------------- %
phi = unwrap(angle(in_bb_int));
% ---------------------------------------------- %
% ETU grid alignment

[in_bb_int_gridAlign, SOC2_int] = reAlignSymbolGrid(y_ref, in_bb_int, fs_int, 1/etu, 0);
SOC2 = round(SOC2_int*fs/fs_int);

% ---------------------------------------------- %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Parameter Determination
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ---------------------------------------------- %
[ISIm, ISId, PR, in_bb_etu] = compute_ISI_param(in_bb_int_gridAlign, y_ref, fc, fs_int, etu, order);
% ---------------------------------------------- %


% ---------------------------------------------- %
[PN_RMS, PN_PKPK] = get_phaseNoiseValue(in_bb_int, SOC2_int, fs_int, etu);
nd_PN_rms = PN_RMS(1)/(EPI*180/pi); % normalized differential phase noise
% ---------------------------------------------- %



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% END %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function [etu, order, PR, Ph, EPI] = get_bit_rate_param(bit_rate, fc)
% function [etu, order, PR, Ph, EPI] = get_bit_rate_param(bit_rate, fc)
%
% DESCRIPTION: Lookup table for the selection of VHBR PSK bit rate related
% parameters.
%
% INPUT:
%       bit_rate...selected bit rate [bits/sec]
%       fc...carrier frequency
% OUPUT:
%       etu...elementary time unit [sec]
```

```
%       order...number of phase levels used for TX of input bit rate
%       PR...phase range [rad]
%       Ph...largest NPV value [rad]
%       EPI...elementary phase interval [rad]
%

if bit_rate == 3*fc/4 ||  bit_rate == 3*fc/2
    Ph = 32/180*pi; % largest NPV value [rad]
    PR = 56/180*pi; % phase range [rad]
    order = 8;
    EPI = 8/180*pi; % [rad]
    if bit_rate == 3*fc/4
        etu = 4/fc;
    else
        etu = 2/fc;
    end
elseif  bit_rate == fc ||  bit_rate == 2*fc
    Ph = 32/180*pi; % largest NPV value [rad]
    PR = 60/180*pi; % phase range [rad]
    order = 16;
    EPI = 4/180*pi; % [rad]
    if bit_rate == fc
        etu = 4/fc;
    else
        etu = 2/fc;
    end
end
```

```
function [SOC_sig, SOC_deg, SOC_plain] = LUT_SOC_vhbrPsk(order)

% function [SOC_sig, SOC_deg, SOC_plain] = LUT_SOC_vhbrPsk(order)
%
% DESCRIPTION: Look-up table for the SOC sequence
%
% INPUT:
%       order...number of PSK levels (modulation order)
% OUPUT:
%       SOC_sig...complex SOC IQ signal at symbol rate
%       SOC_deg...SOC phase signal at symbol rate [degree]
%       SOC_plain...plain SOC data sequence at symbol rate [integer values]
%


switch order
    case 8 % 0 = 32°, 1=24°, 2=16°, 3=8°, 4=0°, 5=-8°, 6=-16°, 7=-24°
        PSK8_cal =  [1; 1; 7; 7; 1; 1; 7; 7; 1; 1; 7; 7; 1; 1; 7; 7; 1; 1; 7; 7; 1; 1; 7; 7; 1; 1; 7;
7;...
                    1; 1; 7; 7; 1; 1; 7; 7; 1; 1; 7; 7; 1; 1; 7; 7];
        PSK8_sync = [1; 7; 1; 7];
        PSK8_tsc =
[0;0;7;3;6;1;5;3;6;2;2;2;7;1;0;3;5;2;3;5;2;3;6;0;7;2;3;3;7;6;4;5;6;1;6;5;2;6;1;3;4;0;2;0;...

6;6;7;0;5;7;3;7;3;0;3;6;6;1;1;0;6;4;0;6;3;5;6;1;1;1;2;6;7;0;7;0;7;3;1;2;4;2;1;5;7;4;0;3;3;2;3;4];
        PSK = [PSK8_cal;PSK8_sync;PSK8_tsc];
        phi_table = [32:-8:-24];

    case  16 % 0 = 32°, 1=28°, 2=24°, 3=20°, 4=16°, 5=12°, 6=8°, 7=4°, 8=0°, 9=-4°, 10=-8°, 11=-12°,
12=-16°, 13=-20°, 14=-24°, 15=-28°
        PSK16_cal =  [1; 1; 15; 15; 1; 1; 15; 15; 1; 1; 15; 15; 1; 1; 15; 15; 1; 1; 15; 15; 1; 1; 15;
15; ...
            1; 1; 15; 15; 1; 1; 15; 15; 1; 1; 15; 15; 1; 1; 15; 15; 1; 1; 15; 15];
        PSK16_sync = [1; 15; 1; 15];
        PSK16_tsc =
[0;0;15;6;11;0;8;4;10;1;0;15;9;13;11;1;4;13;14;2;11;13;3;7;4;10;12;12;4;1;13;15;0;6;...

15;12;5;12;1;4;6;13;0;11;7;7;9;11;4;7;15;6;13;7;12;1;0;6;5;3;14;9;0;12;6;10;11;0;15;14;15;6;...
            15;0;15;0;15;7;2;4;8;3;0;7;11;5;13;2;1;14;15;0];
        PSK = [PSK16_cal;PSK16_sync;PSK16_tsc];
        phi_table = [32:-4:-28];
    otherwise
        disp('No appropriate SOC sequence selected!')
end

SOC_plain = PSK;
```

```
SOC_deg = phi_table(SOC_plain+1);
SOC_deg = [zeros(1,10), SOC_deg];
%calculate complex data
SOC_sig = exp(1i*SOC_deg*pi/180);
SOC_sig= SOC_sig(:);
```

```
function [out_bb] = IQ_demodulation(in, t, fc)
% function [out_bb] = IQ_demodulation(in, t, fc)
%
% DESCRIPTION: Homodyne (IQ) demmodulation of signal 'in'
%
% INPUT:
%       in...RF input signal of length [Nx1]
%       t...sequence of time indices (at sampling rate) [Nx1]
%       fc...carrier frequency
% OUPUT:
%       out_bb...homodyne demodulated input signal
%


out_bb = in.*(exp(-1i*2*pi*fc*t'));
```

```
function [out, f_corr] = derotation(in_bb, idx_unmod, fs, fc)

% function [out, f_corr] = derotation(in_dem, idx_unmod, fs, fc)
%
% DESCRIPTION: performs frequncy error measurement (due to
%              non-synchronuous sampling) and correction
%
% INPUT:
%       in_bb...complex BB signal
%       idx_unmod...index indicating end of non-modulated carrier [samples]
%       fs...sampling frequency [Hz]
%       fc...carrier frequency [Hz]
%
% OUPUT:
%       out...derotated complex BB signal
%       f_corr...measured frequency offset [rad]
%



N = length(in_bb); % signal length
t = [0:1:(N-1)]/fs; t = t(:);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% perform low-pass filtering before frequency error estimation
[b,a] = LP_filter('moving_avg', fs, fc);

in_bb_lp = filter(b,a,real(in_bb)) + 1i*filter(b,a,imag(in_bb));
in_bb_lp = in_bb_lp(length(b)+1:end); % remove filter fade-in transient from signal

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% estimate frequency error

f_norm     = EstimateCarrierFreq(in_bb_lp(1:idx_unmod)');
f_corr = -f_norm*fs;

% derotate signal
out = in_bb.*exp(-1i*2*pi*f_corr*t);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



function FcEst = EstimateCarrierFreq(in_bb_unmod)
% function FcEst = EstimateCarrierFreq(in_bb_unmod)
%
% DESCRIPTION: performs frequncy error measurement
%
% INPUT:
%       in_bb_unmod...non-modulated complex BB signal
%
```

```
% OUPUT:
%       FcEst...measured frequency offset normalize to sampling frequency
%


% 1) rough phase drift estimation
in_bb_unmod             = in_bb_unmod./abs(in_bb_unmod);                        %Totally remove
amplitude fluctuations
IQJumps                 = diff(in_bb_unmod); %phasor speed (instantaneous)

IQjumpsMeanAbs          = mean(abs(IQJumps)); % average of abs of differentiated phasor signal

FcEst                   = asin(IQjumpsMeanAbs/2)/pi;

N1 = length(in_bb_unmod);


derotatedPhasor  = in_bb_unmod.*exp(-1j*2*pi*FcEst*(1:N1));

% 2) fine phase drift estimation

phaseSignalStep2 = unwrap(angle(derotatedPhasor));          %could still wrap around once for extreme
noise conditions

N2 = round(N1/2);
phaseBegin = mean(phaseSignalStep2(1:N2));
phaseEnd   = mean(phaseSignalStep2(end-N2:end));

phaseRot   = phaseEnd-phaseBegin;
frqErr     = phaseRot/2/pi/((N1-1)/2);
FcEst      = FcEst + frqErr;
derotatedPhasor = in_bb_unmod.*exp(-1j*2*pi*FcEst*(1:N1));
```

```
function [b,a] = LP_filter(LP_type, fs, fc)

% function [b,a] = LP_filter(LP_type, fs, fc)
%
% DESCRIPTION: computes filter coefficients for the selected filter.
% Support low-pass filter are: 1) moving average filter ('moving_avg') and
% 2) LP_windowed ('LP_windowed')
%
% INPUT:
%       LP_type...demodulated phase run of BB signal (unmodulated)
%       fs...sampling frequency [Hz]
%       fc...carrier frequency [Hz]
%
% OUPUT:
%       b...filter coef feedforward
%       a...filter coef feedback
%



switch LP_type
    case 'moving_avg'
        N_filt = .5*round(fs/fc); % length to have a Null at 2*fc
        b = 1/N_filt*ones(1, N_filt);
        a = 1;

    case 'LP_windowed'

        %Filtering in TD based on a single windowed sinc lobe
        fco         = 0.67*fc;      %~3dB cut-off frequency; ensure enough roll off (20dB) close to -
fc
        nLobes      = 1.25;         %overshoot control parameter
        [b,LPF_LEN] = LPF_WindowedSinc(fs, fco, nLobes);
        a = 1;

    otherwise
        disp('Unknown method.')
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function [b,LPF_LEN] = LPF_WindowedSinc(fs, fco, nLobes)
% function [b,a] = LP_filter(LP_type, fs, fc)
%
% DESCRIPTION: design of a LP filter with controlled ripple and 3dB
% cutt-off frequency (if first constraint allows)
%
% INPUT:
%       fs...sampling frequency [Hz]
%       fco...cut-off frequency [Hz]
%       nLobes...parameter to conrtol overshoot od stepp response; [1..3],
%       normally <= 2.5
% OUPUT:
%       b...filter coef feedforward
%       LPF_LEN...filter length [samples]
%

sPerCutOff = fs/fco;

if (nLobes >= 20)
    fco = fco*0.99;
elseif (nLobes >=1.0)
    fco = fco*sqrt(2)/nLobes^(1/8); %corr factor to be ~ independent of nLobes
else
    fco = fco*sqrt(2);
end

idxs       = (-ceil(nLobes/2*sPerCutOff):ceil(nLobes/2*sPerCutOff));
LPF_LEN    = length(idxs); %2*round(nLobes/2*sPerCutOff)+1;
filterSinc = sinc(idxs.*(fco/fs*2)).*hanning(LPF_LEN).'; %hanning can reduce 3x the overshoot from 1%
to 0.3%


if(nLobes>1.0)
    %reduce the Gibbs peak in the step response by TD windowing the IR
    windowTD    = blackman(LPF_LEN)'; %BLACKMAN TO REDUCE OVERSHOOT IN STEP RESPONSE
    b = filterSinc.*windowTD;
else
    b = filterSinc;
end

b = real(b);
b = b ./sum(b); %UNIT DC gain
```

```
function [out_bb_gridAlign_m, idxBegin] = reAlignSymbolGrid(y_ref, in_bb_m, fs, fSymb, PLOT)
% function [out_bb_gridAlign_m, idxBegin] =
% reAlignSymbolGrid(y_ref, in_bb_m, fs, fSymb, PLOT)
%
% DESCRIPTION: determine etu grid between SOC reference signal and input
% signal
% Algo STEPS:
%-----------------------------------------------------------------
% STEP A: Align the signals within 1-2 symbols
% STEP B: Align the signals at sample accuracy (symbol grid)
%-----------------------------------------------------------------
% Algo description
% STEP A: Determine crosscorrelation @ symbol rate, using random sampling phase for RX
% STEP B: Reconstruct a pseudo Symbol Response based on sliding xcorr @ symbol rate, and find the
beginning of the rising peak
%-----------------------------------------------------------------
%
% INPUT:
%       y_ref...SOC input reference signal at symbol rate
%       in_bb_m...input complex BB signal
%       fs...sampling frequency [Hz] ==> fs = N*fc  (64>= N >=16)
%       fSymb...symbol frequecy == 1/etu [Hz]
%       PLOT...flag to enable PLOT functionality
%
% OUPUT:
%       out_bb_gridAlign_m...etu grid aligned complex input BB signal
%       idxBegin...start index of first etu
%
```

```
if ~exist('PLOT','var')
    PLOT = 0;
elseif isempty(PLOT)
    PLOT = 0;
end

sps = fs/fSymb; % samples per etu
% find approxiate SOC
offset = 8;
x = unwrap(angle(in_bb_m))*180/pi;
x = x(:);
SOC = find(x>24-offset, 1);
clear x
% remove silence at the beginning, keep 25 etu's silence before first
% modulation
in_bb_m(1:SOC-25*sps) = [];


Nk  = length(y_ref);
Nk2 = floor(length(in_bb_m)/fs*fSymb);
if Nk2 < Nk
    error('Measured data is shorter than reference data')
end

MaxSymbMisalign = Nk2-Nk;   % Maximum misalignment in etu's between 2 signal inputs

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% (Step A) Realign on a symbol level:
% Input             : y_ref,                      Reference symbols
% Input             : in_bb_m                         resampled (to int fc) demod data (scope into
complex Baseband), LONGER than REF
% Output(internal)  : BasebandRx_SymbAlign_m               ALIGNED to REF ± 2 Symbols (trimmed
signal)             fs = N*fc
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
y_ref_m  = genericOversample(y_ref,fs/fSymb); %Oversampled REFERENCE signal @ target Sample Rate

% time vectors in REFERENCE time continuous signal domain (alignment output domain)
Ns_m            = length(y_ref_m);
timeVecfsInt_m  = (0: Ns_m-1);
timeVecEtu_m    = timeVecfsInt_m/fs*fSymb; %for PLOTs

% (1) take a initial random sampling phase
%-------------------------------------------
SymbolsRx_k     = in_bb_m(sps:sps:end); %any random sampling offset will do here

% (2) Align on a symbol level
%-------------------------------------------
delay2          = find_Delay_1_Within_2(y_ref, SymbolsRx_k, MaxSymbMisalign, 0);

% Internal Output: extract  symbol level aligned @ sample rate, ±2 symbols
idxBegin        = delay2*sps;
idxEnd          = (delay2+Nk)*sps-1;
BasebandRx_SymbAlign_m = in_bb_m(idxBegin:idxEnd);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% (Step B) Realign on a sample level:
% Input: y_ref                      REF TX symbols                      fSymb
% Input: BasebandRx_SymbAlign_m:            ALIGNED to REF ± 2 Symbol           fsOut = N*fc
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% STEP B Into larger detail:
%
% (1) Get a pseudo symbol-response (signature1) through sliding cross correlation @ symbol rate
%     Find the present RX start being correlated (xc) with future TX which indicates too late
sampling
%     Sliding over RX, a peak will be found
% (2) Find the peak in signature1
% (3) Consider signature2 the slope (derivative) of signature1.
% (4) identify flat region in signature 2, to the left of the peak in signature1
% (5) define "end of flat region", or corner point of signature 2
% (6) Reference Timing is @ the end of this flat region, corrected for the derivative delay and a
guard interval.

MAX_MISALIGN        = 2;        %upper bound to input misalignment
FLAT_REGION_MIN     = 2;        %algo specific, MIN 1, 2 safer; get at least a full etu leading the
beginning of symbol response
```

```
GuardTimeInterval   = 2;        % This param depends loosely on below def of MaxDev and on input SNR
                                % These vals are taken for SNR_dB_IQ=30dB (very noisy), which tend to
increase guard interval.


BasebandRx_SymbAlign_m = BasebandRx_SymbAlign_m./max(abs(BasebandRx_SymbAlign_m)); %energy scaling to
unit circle

% (1) Get a pseudo symbol-response (signature1)
% ---------------------------------------------
Ns_m                        = length(BasebandRx_SymbAlign_m);
allTimes                    = (1:Ns_m);
gridVals                    = (1:(2*MAX_MISALIGN+FLAT_REGION_MIN+1)*sps);
futureXC                    = zeros(1,length(gridVals));
for iGrid = gridVals
    symbolGrid              = allTimes(iGrid:sps:Ns_m);
    BBout_try_k             = BasebandRx_SymbAlign_m(symbolGrid);
    SymbolsPSKTx_sl_k       = y_ref(1:length(BBout_try_k)); %make it same length

    x1 = BBout_try_k(1:end-MAX_MISALIGN-FLAT_REGION_MIN);               %present rx
    x2 = SymbolsPSKTx_sl_k(1+FLAT_REGION_MIN+MAX_MISALIGN:end);         %future  tx symbols
    Lstart  = 60;  %trim out periodic part (44+silence), need white signal here
    x1 = x1(Lstart:end);
    x2 = x2(Lstart:end);
    futureXC(iGrid)         = sum((x1-mean(x1)).*conj((x2-mean(x1))));      %cross covariance
end

DelaySignature = real(futureXC);

% (2) Find the peak in signature1
% -------------------------------------------
peakValRef                  = abs(xcov(y_ref,y_ref,0));
[peakVal,peakPos]           = max(DelaySignature);
if (peakVal < peakValRef/4)
    error('Symbol Grid Alignment Failed, no reliable xcorr peak found');
end



% (3) Consider signature2 the slope (derivative) of signature1.
% -------------------------------------------
slopeEst                    = [1,3,2,-2,-3,-1];                     %a slope estimator robust to
noise, reduces randomness of symbol grid by a gppd deal (try [1,-1] to see)
slopeEst                    = slopeEst./sum(abs(slopeEst));
lenSE                       = length(slopeEst);
slopeDelay                  = max(1,floor((lenSE-1)/2));
DelaySignatureDiff          = filter(slopeEst,1,DelaySignature);
DelaySignatureDiff(1:lenSE) = DelaySignatureDiff(1+lenSE);          %remove transient



% (4) identify flat region in signature 2, to the left of the peak in signature1
% -------------------------------------------
% flat region first guess based on signature 1 peak
flatRegionCornerPosMin      = peakPos - round(1.3*sps)  ;    %point P1, earliest corner position
idx1                        = max(lenSE,flatRegionCornerPosMin -sps);
flatRegionDiff              = DelaySignatureDiff(idx1:flatRegionCornerPosMin); %this is 1 etu of flat
region, to the left of P1

% If needed adjust flat region by 1/2 etu to the right (if we detect we are too early (EMC case),
this enables calculating MaxDev on a more significant portion of the flat region)
flatRegionDiffCandidate2= DelaySignatureDiff(idx1+sps/2:flatRegionCornerPosMin+sps/2);
if sqrt(var(flatRegionDiffCandidate2)) < 2*sqrt(var(flatRegionDiff))
    fprintf('Overshooting Channel suspected, improving flat region location\n');
    flatRegionCornerPosMin  = flatRegionCornerPosMin + sps/2;
    idx1                    = max(lenSE,flatRegionCornerPosMin -sps);
    flatRegionDiff          = flatRegionDiffCandidate2;
end

% (5) define "end of flat region", or corner point of signature 2
% -------------------------------------------
% define "end of flat region", by a vertical interval within which flat region TO THE RIGHT P1 should
be contained
MaxDev                      = 6*sqrt(var(flatRegionDiff));
```

```
MeanVal                     = mean(flatRegionDiff);



% (6) Reference Timing is @ the end of this flat region, corrected for the derivative delay and a
guard interval.
% ------------------------------------------------------------
idxPosFirstGuess =  flatRegionCornerPosMin + find(DelaySignatureDiff(1+flatRegionCornerPosMin:end) >
MeanVal+MaxDev,1,'first');
if isempty(idxPosFirstGuess)
    idxPosFirstGuess = 0;
end
backOff           = GuardTimeInterval + slopeDelay; %(2..4)
idxSymbolBegin    = idxPosFirstGuess - backOff;



% extract (re-Index) the sample aligned within input signal
idxBegin = (delay2 - FLAT_REGION_MIN - MAX_MISALIGN)*sps + idxSymbolBegin;
idxEnd   = idxBegin + Nk*sps - 1;
out_bb_gridAlign_m = in_bb_m(idxBegin:idxEnd);
```

```
function y = genericOversample(x,OSF)
% function y = genericOversample(x,OSF)
%
% DESCRIPTION: perfrom resampling using nearest neighbor interpolation
%
% INPUT:
%       x...input data sequence
%       OSF...oversampling factor, the ration between target and actual
%       sampling rate
%
% OUPUT:
%       y...resampled input data sequence
%

outLen = floor(length(x)*OSF);
idx_y = (1:outLen);
%t_idx = round(idx_y/OSF);
t_idx = ceil(idx_y/OSF);
y     = x(t_idx);
```

```
function [delay2,signal2Aligned] = find_Delay_1_Within_2(signal1,signal2, MaxMisalign, DEBUG)
% function function [delay2,signal2Aligned] =
% find_Delay_1_Within_2(signal1,signal2, MaxMisalign, DEBUG)
%
% DESCRIPTION: perfrom alignment of reference signal and an observed signal
%   maximizing a correlation cost function
%
% INPUT:
%       signal1...reference input signal
%       signal2...not aligned observed signal
%       MaxMisalign...Maximum allowed misalignment between the 2 input
%       signals [etu]
%       DEBUG...flag to enable PLOT functionality
%
% OUPUT:
%       delay2...determined delay
%       signal2Aligned...delay2 aligned signal2
%

signal1 = signal1(:);
signal2 = signal2(:);
if ~exist('DEBUG','var')
    DEBUG = 0;
end

%check if signal2 > signal1
N1   = length(signal1);
N2   = length(signal2);
if abs(N1-N2) > MaxMisalign
```

```
    error('Signal lengths differ more than maximum misalignment. Wrong signals?');
end

if (N1 ~= N2)
    if (N1 > N2)
        error('find_Delay_1_Within_2: signal 2 shorter than signal 1, not supported!');
    else
        %pad 1 to match 2 in length (makes it easier to interprete xcorr peak as delay)
        nPad = abs((N2-N1));
        if(mod(nPad,2))
            %to avoid troubles make the length differ by even number
            signal2 = signal2(1:end-1);
            N2      = N2-1;
            nPad    = nPad-1;
        end
        nPadFront   = nPad/2;
    end
else
    nPadFront = 0;
end
signal1Pad  = [zeros(nPadFront,1);signal1;zeros(nPadFront,1)];


[corrVal,lags]      = xcov(signal1Pad,signal2, MaxMisalign);
peakVal             = max(abs(corrVal));

peakValRef1         = abs(xcov(signal1,signal1, 0));
peakValRef2         = abs(xcov(signal2,signal2, 0));
if peakVal< 0.25 * sqrt(peakValRef1*peakValRef2)
    warning('find_Delay_1_Within_2: signals quite different... is there a mistake somewhere?');
end


idxPeak             = find(abs(corrVal)== max(abs(corrVal)));
lagPeak             = lags(idxPeak);                        %time align reference

if (length(idxPeak)>1)
    warning('find_Delay_1_Within_2: Unclear peak... is there a mistake somewhere?');
    DEBUG           = 1;
    idxPeak         = idxPeak(1);
end


% Apply time alignment on 2
%example of how time alignment works
%                                   |                     central Lag
%                   012 3456789012345678901 2345678     rx len = 29
%                   qrs abcdefghijklmnopqrs abcdefg
%RX                 ---|---------.---------|-------
%TX                 00000|---------.---------|00000     tx len = 19
%                             ^ ^
%Corr Peak                   -2                        lagPeak = -2
%                  |-->|                              offsetStart = 3 = (29+1)/2 - (19+1)/2 + lagPeak


delay2 = -lagPeak + nPadFront; %correct answer
if (delay2>=0)
     signal2Aligned  = signal2(1+delay2:end);    %cut front
     signal2Aligned  = signal2Aligned(1:N1);
else
     delay2
     error('signal 2 does NOT contain signal 1');
end
```

```
function [ISIm, ISId, PR, out_bb_etu] = compute_ISI_param(in_bb_int, y_ref, fc, fs_m, etu, order)

% function [ISIm, ISId, PR, IQ_symbs] =
% compute_ISI_param(in_bb_int, y_ref, fc, fs_m, etu, order)
%
% DESCRIPTION: ISI parameter determination. 1) compute pseudo step response
% by solving the system idetification problem using the LLS method. 2)
% Compute the ISI parameter from pseudo step response
%
% INPUT:
%       in_bb_int...grid aligned complex input BB signal
```

```
%       y_ref...complex reference SOC signal
%       fc...cariier frequency [Hz]
%       fs_m..sampling frequency [Hz] ==> N*fc N>= 16
%       etu...elementary time unit [sec]
%       order...number of PSK levels (modulation order)
% OUPUT:
%       ISIm...measured ISI magnitde [EPI]
%       ISId...measured ISI rotation [degree]
%       PR...measured phase range [degree]
%       out_bb_etu...grid aligned signal at symbol rate (sampled at ISIm, ISId instant)
%

%#########################################################################################
%System Identification (Channel Estimation) :
%Input: SymbolsPSKTx_k              fSymb
%Input: in_bb_int     N*fc
%Input: fs_p                        estimation rate >= 4*fc
%Input: Ntaps_k                     Nr of symbols (incl present) in estimate output
%#########################################################################################

%get a row vectors
y_ref = y_ref(:).';

%find the minimum and maximum constellation points
[const_min] = min(angle(y_ref));
[const_max] = max(angle(y_ref));
%calculate phase range
PR_deg = (const_max - const_min)*180/pi;

fSymb = 1/etu;


Ntaps_k          = 4; %# symbols
sps_m    = fs_m*etu;
%step1: downsample to 4*fc (_p) (first LPF if signal has not been LPF else noise will alias in band)
%==============================
fs_p        = 16*fc;
sps_p       = fs_p*etu;
dwnspl_m2p  = fs_m/fs_p;

BasebandOut_GridAlign_p = in_bb_int(dwnspl_m2p:dwnspl_m2p:end);

% Step2:  Now in a loop estimate directly the Symbol Response @ 4fc sliding over one symbol
%=============================================================================================================
Ntaps_p             = Ntaps_k*sps_p;
EstSymbRsp_Slide_p  = zeros(1,Ntaps_p);

for ii=1:sps_p
    % downsample measured signal to one sample per etu ==> shift per 1
    % sample each
    BasebandOut_GridAlign_k          = BasebandOut_GridAlign_p(ii:sps_p:end);
    % system identification: step wise computation of pseudo step response
    [hVec_k, RMSerr] = LLS_FIR_Cplx_Channel_Fit_NTaps(y_ref, BasebandOut_GridAlign_k.', Ntaps_k);
    EstSymbRsp_Slide_p(ii:sps_p:end) = hVec_k;
end

%#########################################################################################
% ISI parameter Estimation : observation interval == last carier period of
% each etu
%#########################################################################################
spc_p = fs_p/(fc);
for ii=0:spc_p
    jj = spc_p-ii+1;
    %extract symbol response for each sampling instant of the last carrier
    %period of an etu
    EstSymbRsp_k = EstSymbRsp_Slide_p(sps_p-ii:sps_p:end);

    [ISIm_SI(jj), ISId_DEG(jj)] = ISIparams_From_SymbRsp_k(EstSymbRsp_k,PR_deg,order, fc,1/etu);
end

% select maximum ISIm in last carrier period
[v, IDX] = max(ISIm_SI);

ISIm = ISIm_SI(IDX);
ISId = ISId_DEG(IDX);
```

```
ds_fact = fs_p*etu;
symStrtIdx = sps_p-IDX+1;
out_bb_etu = BasebandOut_GridAlign_p(symStrtIdx:ds_fact:end);

PR = (max(angle(out_bb_etu))-min(angle(out_bb_etu)))*180/pi; %Measure PR
% segment (approx)
```

```
function [hVec, RMSerr] = LLS_FIR_Cplx_Channel_Fit_NTaps(y_ref, in_bb_etu, Ntaps)
% function [hVec, RMSerr] = LLS_FIR_Cplx_Channel_Fit_NTaps(y_ref,
% in_bb_etu, Ntaps)
%
% DESCRIPTION: solves system indetification probelm by estimation the channel using LLS method
%
% INPUT:
%       y_ref...complex reference signal at etu rate [1 sample/etu]
%       in_bb_etu...observe complex signal at etu rate [1 sample/etu]
%       Ntaps...estimate first Ntaps coefficients
%
% OUPUT:
%       hVec...estimated filter coefficients
%       RMSerr...error between filtered reference and observed signal
%



%    [ x(k)   x(k-1) ...  x(k-Ntaps+1)        ]  [h0          ]          [y0]
%    [ x(k-1) x(k-2) ...                      ]  [h1          ]          [  ]
%    [ .      .                               ]  [..          ]     ~=   [  ]
%    [ .      .                               ]  [..          ]          [  ]
%    [ x(k-L) x(..)  ...  x(k-Ntaps+1-L)      ]  [h(Ntaps-1)  ]          [yL]

%x = input, apriori, noiseless      TX
%y = output, observation, noisy     RX

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Estimate channel in the form [h0,h1,h(Ntap-1)] using LLS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if(Ntaps>length(y_ref))
    error('Nr taps larger than input size');
end
if(length(y_ref)~=length(in_bb_etu))
    error('Input sizes don''t match');
end


%prepare data
m       = length(y_ref)-Ntaps+1;  %Column size ~ observation Length
X       = zeros(m,Ntaps);
for iCol = 0:Ntaps-1
    X(:,iCol+1) = y_ref(end-iCol  : -1 : Ntaps-iCol)';      %(m x Ntaps) matrix (a-priori, noiseless)
end
y       = in_bb_etu(end:-1:Ntaps)';                             %(m x 1)     vector (observations,
noisy)

%Normal Equation : X^T.X.h - X^T.y = 0 <=> h = MinArg(|X.h-y|^2, for all h)
XT      = X';
XTX     = XT*X;         % Symmetric and positive definite, noiseless
XTy     = XT*y;

if(cond(XTX)>1.2e8)
    fprintf('\n WARNING : Ill conditioned fitting may result ## !!\n');
end

%Solve linear system XTX*h = XTy
hVec = linsolve(XTX,XTy);

hVec = conj(hVec)';

RMSerr = sqrt(mean((abs(X*hVec'-y)).^2));
```

```
function [ISIm_SI, ISId_DEG] = ISIparams_From_SymbRsp_k( SymbRsp_k, PR,order, fc, fSymb)

% function [ISIm_SI, ISId_DEG] = ISIparams_From_SymbRsp_k( SymbRsp_k, PR,order, fc, fSymb)
```

```
%
% DESCRIPTION: compute the ISI_m and ISId parameters from the system
% identification coefficients
%
% INPUT:
%   SymbRsp_k...system identification coefficients
%   PR...phase range [degree]
%   order...number of PSK levels (modulation order)
%   fc...carrier frequency [Hz]
%   fSymb...symbol frequecy == 1/etu [Hz]
%
% OUPUT:
%       ISIm...measured ISI magnitde [EPI]
%       ISId...measured ISI rotation [degree]
%


EPI = PR/(order-1); % [degree]
h0         = SymbRsp_k(1);
h1         = SymbRsp_k(2);

% Estimate main ISI parameters

ISIratio                = sum(abs(SymbRsp_k(2:end)))   / abs(sum(SymbRsp_k)); %SAME AS USED IN PSK
RECEIVER PERFORMANCE LOOPS

%ISIangle = asin(L/R), where L=distance btw 2 outer pts in the cloud
%L/2=r*sin(Dphi/2) from the ISI cloud triangle -> L/R = r/R * 2*sin(Dphi/2)
ISIangle               = asin(2*ISIratio*sin(PR/2/180*pi)); %size of ISI cloud in Degrees, for 0
detuning
ISIangleDEG            = ISIangle*180/pi;
ISIm_SI                = ISIangleDEG/EPI;
ISId_DEG               = -(angle(h1)-angle(h0))*180/pi;
% perfom scaling of ISIm parameter to compensate for LP-filtering in
% ananlysis tool
if fSymb == fc/4
    ISIm_SI = 0.9*ISIm_SI;
else
    ISIm_SI = 0.8*ISIm_SI;
end
```

```
function [PN_rms] = get_phaseNoiseValue(x, SOF,fs, etu)

% function [PN_RMS, PN_PKPK] = get_phaseNoiseValue(x, SOF, fs, etu)
%
% Description: computes the differential phase noise value of an unmodulated carrier
% according to the definition for VHBR PSK of ISO/IEC 14443-2
%
% INPUT:
%       x...complex IQ input signal
%       SOF...sample index of first modulated symbol
%       fs...sampling frequency
%       etu...symbol duration [sec]
% OUPUT:
%       PN_rms...root-mean-square differential phase noise [degree]
%


%calculate downsample factor
fact = fs*etu;
%downsample the signal taking SOF oversampling factor and number of symbols
%into accout
unmod = (floor(SOF/fact)-1);
IQ_symbs = x(round(SOF-unmod*fact:fact:unmod*fact));
phi = angle(IQ_symbs)*180/pi;
phi_err = diff(phi);

% compute the differential phase error

%calculate the maximum phase error
PN_pkpk = max(phi_err)-min(phi_err);

%calculate the rms differential phase error
```

```
PN_rms = sqrt(mean(diff(phi).^2));
```

J.3.12 Example test report (informative)

Below, an example test report for one test position in the operating volume and the bit rate of *fc* is shown.

EXAMPLE 1: PASS case

```
PCD transmission test
------------
General Setting:
Bit Rate: 13,56 Mbit/s
Position (x,y,z):  (0,00 mm, 0,00 mm, 37,50 mm)
------------
Setup Description [optionally provide additional information]:

------------
Measurement Results
Phase range (PR)                      = 60,23 °

ISI Measures
ISI magnitude (ISIm)                  = 0,46
ISI rotation (ISId)                   = -57,61 °

Phase Noise Measure
Normalized differential phase noise   = 0,0298
------------
PASS-FAIL summary report
Phase range (PR)                      : PASS
ISI magnitude (ISIm)                  : PASS
Normalized differential phase noise   : PASS
------------
Overall test result                   : PASS
```

EXAMPLE 2: FAIL case (due to noise)

```
PCD transmission test
------------
General Setting
Bit Rate: 13,56 Mbit/s
Position (x,y,z):  (0,00 mm, 0,00 mm, 37,50 mm)
------------
Setup Description [optionally provide additional information]:

------------
Measurement Results
Phase range (PR) (measured)           = 60,99 °

ISI Measures
ISI magnitude (ISIm)                  = 0,46
ISI rotation (ISId)                   = -58,05 °

Phase Noise Measure
Normalized differential phase noise   = 0,0785
------------
PASS-FAIL summary report
Phase range (PR)                      : PASS
ISI magnitude (ISIm)                  : PASS
Normalized differential phase noise   : FAIL
------------
Overall test result                   : FAIL
```

EXAMPLE 3: FAIL case (due to ISI_m)

```
PCD transmission test
------------
General Setting
Bit Rate: 13,56 Mbit/s
Position (x,y,z):  (0,00 mm, 0,00 mm, 37,50 mm)
------------
```

```
Setup Description [optionally provide additional information]:

------------
Measurement Results
Phase range (PR)                      = 60,22 °

ISI Measures
ISI magnitude (ISIm)                  = 1,70
ISI rotation (ISId)                   = -50,03 °

Phase Noise Measure
Normalized differential phase noise   = 0,0001
------------
PASS-FAIL summary report
Phase range (PR)                      : PASS
ISI magnitude (ISIm)                  : FAIL
Normalized differential phase noise   : PASS
------------
Overall test result                   : FAIL
```

J.4 PCD signal creation for PICC reception tests (informative)

J.4.1 Overview/Introduction

The following clauses describe how to create test signals for PICC reception tests as required for conditions 1 to 5 of J.2.2.4. Test signals are digitally pre-conditioned before transmission.

J.4.2 ISI_m and ISI_d test signal creation

The test signals are created using the baseband model of the test PCD antenna with impedance matching network for bit rates higher than *fc*/128. This baseband model can be used to derive a transfer function H_{bb} which describes the physical antenna resonator. The test signal is created by the following steps:

1) The sequence of NPVs in its complex representation is filtered by H_{bb}

2) Upconversion of the filtered NPVs using the carrier signal

The resulting digital signal is equivalent to the signal observed at the air interface. Therefore, the transmission of this signal would result in slightly different ISI_m and ISI_d parameters due to the additional filter-effect of the test PCD antenna used for transmission.

This signal description does not take into account the additional filter-effect of the test PCD antenna.

A time-discrete baseband filter that creates the desired Inter-symbol interference signal (with given ISI_m and ISI_d) is described in the z-domain by:

$$H_{bb}(z) = (1-p)/(1-p \cdot z^{-1}),$$

with *p* the complex pole. The complex pole position can be computed for desired ISI_m and ISI_d parameters according to

$$p = \left(\frac{\frac{1}{2}\sin(ISI_m \cdot EPI) \cdot \exp(j \cdot ISI_d)}{\sin(\frac{1}{2} \cdot PR)} \right)^{\left(\frac{T_{sr}}{etu - 1/fc}\right)},$$

where *j* is the imaginary unit and T_{sr} is the sample duration (the inverse of the sample rate). This equation is only valid if $T_{sr} \leq 1/fc$.

J.4.3 Normalized differential phase noise test signal creation

The defined test signal is created by adding a low-pass filtered pseudo random white noise to the sequence of NPVs in its complex representation. A 2nd order, Butterworth type low-pass filter with cut-off frequency of 100 kHz is used to filter frequency components above the cut-off frequency. The filter characteristic is illustrated in Figure J.6.

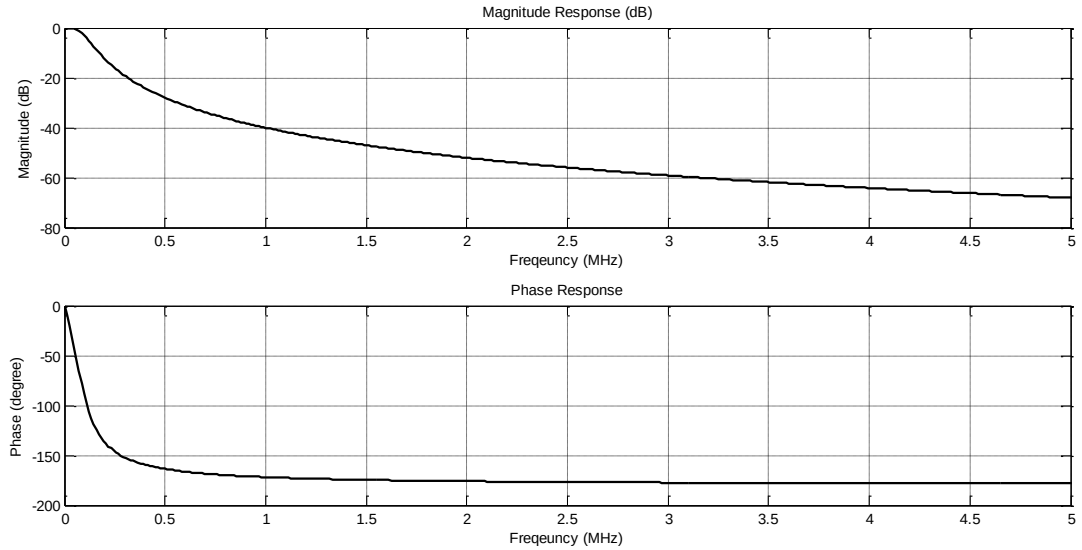

Figure J.6 — Filter characteristics of the 2nd order Butterworth filter

J.4.4 PCD signal creation tool

This informative MATLAB-code describes how the five test conditions in the digital base-band domain could be created digitally for bit rates of 3*fc*/4, *fc*, 3*fc*/2 and 2*fc* from PCD to PICC. The implementation consists of a single file. Select one of the five test conditions and one of the 4 bit rates.

```
% main signal generation function for bit rates of 3fc/4, fc, 3fc/2 and 2fc
% from PCD-PICC
% Signal creation function for PICC reception tests
function Sout = main()

COND = 'condition1'; % enter one of the following test cond: {'condition1',
% 'condition2', 'condition3', 'condition4', 'condition5'}

%--------------------------------------------------------------------------
% definition of constants

fc = 13.56e6; % carrier frequency
bit_rate = 3*fc/4; % specify bit rate
numOfNPV = 512; % number of NPVs
% Condition 1
d_pn_rms = 0.035; % maximum normalized differential phase noise (PICC RX)[EPI]
% Condition 2
ISIm_c2 = 1.1; % [EPI]
ISId_c2 = 45; % [degree]
% Condition 3
ISIm_c3 = 1.1; % [EPI]
ISId_c3 = -45; % [degree]
% Condition 4
ISIm_c4 = 0.6; % [EPI]
ISId_c4 = 120; % [degree]]
% Condition 5
ISIm_c5 = 1.6; % [EPI]
ISId_c5 = 0; % [degree]


if bit_rate == 3*fc/4 ||  bit_rate == 3*fc/2
    Ph = 32/180*pi; % largest NPV value
    PR = 56/180*pi; % phase range
    order = 8;
    if bit_rate == 3*fc/4
        etu = 4/fc;
    else
        etu = 2/fc;
    end
elseif  bit_rate == fc ||  bit_rate == 2*fc
    Ph = 32/180*pi; % largest NPV value
    PR = 60/180*pi; % phase range
    order = 16;
    if bit_rate == fc
        etu = 4/fc;
    else
        etu = 2/fc;
    end
end
Tsr = etu; % sample duration or the inverse sample rate
%--------------------------------------------------------------------------

EPI = PR/(order-1); % compute EPI
NPV_table = [Ph:-EPI:(Ph-(order-1)*EPI)]; % Compute all possible NPVs

% randon signal creation (sequence of NPVs)
Sin_int = randint(numOfNPV,1,order); % sequence of NPV in integer value representation
Sin_phase = NPV_table(Sin_int+1); % corresponding phase signal [rad]
Sin_phase = Sin_phase(:);
Sin = exp(1i*Sin_phase); % complex representation of the sequence of NPVs

switch COND
    case 'condition1'
```

```
        noise = awgnPhaseNoise(numOfNPV, EPI, d_pn_rms, 1/Tsr);
        Sout = Sin.*noise;

    case 'condition2'
        [b,a] = ISI_pole(ISIm_c2, ISId_c2, PR, EPI, Tsr, etu);
        Sin_ = [ones(2,1); Sin]; % add zeros for correct initialization
        Sout = filter(b,a,Sin_);
        Sout = Sout(3:end); % remove leading zeros for initialization

    case 'condition3'
        [b,a] = ISI_pole(ISIm_c3, ISId_c3, PR, EPI, Tsr, etu);
        Sin_ = [ones(2,1); Sin]; % add zeros for correct initialization
        Sout = filter(b,a,Sin_);
        Sout = Sout(3:end); % remove leading zeros for initialization
    case 'condition4'
        [b,a] = ISI_pole(ISIm_c4, ISId_c4, PR, EPI, Tsr, etu);
        Sin_ = [ones(2,1); Sin]; % add zeros for correct initialization
        Sout = filter(b,a,Sin_);
        Sout = Sout(3:end); % remove leading zeros for initialization
    case 'condition5'
        [b,a] = ISI_pole(ISIm_c5, ISId_c5, PR, EPI, Tsr, etu);
        Sin_ = [ones(2,1); Sin]; % add zeros for correct initialization
        Sout = filter(b,a,Sin_);
        Sout = Sout(3:end); % remove leading zeros for initialization
    otherwise
        disp(sprintf('No valid test condition selected.'))
        return
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PLOT functionality
% PLOT circle segment
phi = [45:-1:-35]*pi/180;
r = ones(1,length(phi));
x = sqrt(r.^2./(1+(atan(phi).^2)));
y = x.*atan(phi);
% scatter PLOT of reference signal and conditioned signal
figure, scatter(real(Sin), imag(Sin), 'rs', 'linewidth', 2)
hold on, scatter(real(Sout), imag(Sout), 'g')
hold on, PLOT(x,y, 'k--')
legend('NPVs', 'APVs', 'Location','NorthWest')
set(gca, 'box', 'on')
xlim([0 1.1])
ylim([-.5 .7])


%-------------------------------------------------------------------------
function out = awgnPhaseNoise(SinLen, EPI, d_pn_rms, fs)
% returns LP-filtered AWG noise normalized to d_pn_rms
% INPUT:
% SinLen...required vector length of noise
% EPI...elementary phase interval [rad]
% d_pn_rms...target normalized differential phase noise [rms]
% fs...sample rate
% OUTPUT:
% out...phase noise in its complex representation

noise = randn(SinLen*10,1);
noise = noise(end-SinLen+1:end);
[b,a] = butter(2, 100e3/(fs/2));
out = filter(b,a, noise); % low-pass filtering of noise
% adapt noise valule to target value (d_pn_rms)
d_pn = sqrt(mean(diff(out).^2))/EPI;
mult_fac = d_pn_rms/d_pn;
out = out*mult_fac;

out = exp(1i*out); % complex representation of the phase noise
%-------------------------------------------------------------------------
function [b,a] = ISI_pole(ISIm, ISId, PR, EPI, Tsr, etu, fc)
% computes ISI filter coefficients
% INPUT:
% ISIm...Inter-symbol interference magnitude [EPI]
% ISId...Inter-symbol interference rotation [degrees]
% EPI...elementary phase interval [rad]
% Tsr...sample duration [sec]
% etu...elementary time unit
% fc...carrier frequency [Hz]
```

```
% OUTPUT:
% b...feed forward filter coefficient
% a...feedback filter coefficients
Tc = 1/fc;
p = (1/2*sin(ISIm*EPI)*exp(1i*ISId/180*pi) / sin(1/2*PR))^(Tsr/(etu-Tc));
b = (1 - p);
a = [1, -p];
```