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

Document type: Text for CD ballot or comment

Title: Notification of Ballot: ISO/IEC 10373-6:2011/PDAM 5 - Identification cards - Test methods - Part 6: Proximity cards - AMENDMENT 5 Bit rates of 3fc/4 and fc

Status:

WORK ITEM: 60115

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

Date of document: 2011-07-14

Expected action: VOTE

Action due date: 2011-09-15

No. of pages: 19

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 N 1815

Date: 2011-04-01

ISO/IEC 10373-6:2011/PDAM 5

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 and *fc*

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

AMENDEMENT 5
Débits binaires supérieurs à fc/2 jusqu'à fc

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

C:\Userdata\Standards\10373-6\Amendment 4 VHBR\Editing Ispra 2011\Send to WG8 member review\wg8nxxxx_CD10373-6_AMD5_VHBR_PSK_V3.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

Amendment 5: Bit rates of 3*fc*/4 and *fc*

Page 18 of ISO/IEC 10373-6:2011

Add following new sub clause at the end of 7.1.5.3:

"

7.1.6 VHBR PSK IQ segment and waveform characteristics

7.1.6.1 Purpose

This test determines the IQ segment transmitted by the PCD PSK signal as well as the noise and inter-symbol interference parameters as defined in ISO/IEC14443-2:2010 PDAM4.

7.1.6.2 Test procedure

a) Position the calibration coil at an arbitrary position in the defined operating volume and display the induced coil voltage on a suitable oscilloscope (see 5.1.1). The sampling scope should capture at least the full first frame PCD to PICC communication after the switch to the data rate under test has occurred. The activation of the PSK data rate is performed as defined in ISO/IEC 14443-4. Determine the waveform characteristics using the analysis tool defined in Annex Amd.5 A and B.

b) Tune the Reference PICC to 16,5 MHz as described in 5.4.3 steps a) to g) and switch the jumper J1 to position 'c'.

c) Place the Reference PICC at a particular position in the PCD operating volume.

d) Apply and adjust a DC voltage at CON2 to obtain a DC voltage at connector CON3 of V_{Load}.

NOTE 1 If a DC voltage of V_{Load} cannot be reached at the selected position, the maximum achievable voltage should be used for the test.

e) If the unmodulated voltage on CON4, measured with a suitable oscilloscope (see 5.1.1) is below 1 V (peak-to-peak), use an alternative pick up coil to determine the waveform characteristic.

NOTE 2 This alternative pick up coil should have a "figure of 8" shape with 15 mm radius positioned farthest away from the Reference PICC to minimize coupling and as close as possible to the PCD antenna to maximize induced voltage.

f) Determine the waveform characteristics from the voltage at CON4 or at the alternative pick up coil using the analysis tool defined in Annex Amd.5 A. For this purpose the PCD shall transmit an I-Block with predetermined random data of 1kByte in the INF-field using an extended frame as defined in ISO/IEC 14443-3.

g) Repeat steps c) to f) for various positions within the operating volume.

NOTE 3 The selected position of the calibration coil within the operating volume is not expected to affect the results.

NOTE 4 The Reference PICC load does not represent the worst case loading effect of a PICC. Higher loading effects may be achieved with resonance frequencies closer to carrier frequency (e.g. 15 MHz or 13,56 MHz).

7.1.6.3 Test report

The measured voltage at point f) in section 7.1.6.2 should be sampled and analyzed with the analysis tool reported in Annex Amd.5 B.

The test report shall give the measured PSK IQ segment, the ISI parameters (magnitude and rotation) and the RMS phase noise of the PCD field, within the defined operating volume in unloaded and loaded conditions. The measured parameters shall be within the limits specified in ISO/IEC 14443-2:2010 Amd.4.

"

Page 22 of ISO/IEC 10373-6:2011

Add following new sub clause at the end of 7.2.4.5:

"

7.2.5 PICC VHBR PSK reception Test

7.2.5.1 Purpose

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

7.2.5.2 Test conditions

Five test conditions are defined with noise and ISI levels at the border of the signal parameters as defined in 14443-2:2010 Amd.4. The test conditions are created using the test PCD assembly for bitrates higher than *fc*/128 in combination with digital pre-conditioning of the transmitted symbols as defined in Annex Amd.5 C and D.

— Condition 1: Pseudo-random white noise is added to the transmitted symbols such that the root-mean square (RMS) phase noise is the maximum value as defined in ISO/IEC 14443-2:2010 Amd.4.

— Condition 2: The test PCD signal is digitally pre-conditioned to have a maximum inter-symbol interference for a detuning angle ≤ $ISI_{d,lim}$. The detuning angle is set at $ISI_{d,lim}$ degrees as defined in ISO/IEC 14443-2:2010 Amd.4.

— Condition 3: The test PCD signal is digitally pre-conditioned to have a maximum inter-symbol interference for a detuning angle ≤ $ISI_{d,lim}$. The detuning angle is set at - $ISI_{d,lim}$ degrees as defined in ISO/IEC 14443-2:2010 Amd.4.

— Condition 4: The test PCD signal is digitally pre-conditioned to have a maximum inter-symbol interference for a detuning angle > $ISI_{d,lim}$. The detuning angle is set at 60 degrees.

— Condition 5: The test PCD signal is digitally pre-conditioned to have a maximum inter-symbol interference for a detuning angle > $ISI_{d,lim}$. The detuning angle is set at - 60 degrees.

NOTE 1 These conditions are applied after switching to the bit rate under test.

NOTE 2 Annex Amd.5 C and Annex Amd.5 D informatively describe 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}.

7.2.5.3 Test procedure

A PICC supporting the optional 3*fc*/4 bit rate shall operate under the defined conditions after selection of a bit rate of 3fc/4. This PICC shall respond correctly to an I-block transmitted at a bit rate of 3*fc*/4. The activation of the bit rate uses S(PARAMETER) mechanism as defined in ISO/IEC 14443-4.

A PICC supporting the optional *fc* bit rate shall operate under the defined conditions after selection of a bit rate of *fc*. This PICC shall respond correctly to an I-block transmitted at a bit rate of *fc*. The activation of the bit rate uses S(PARAMETER) mechanism as defined in ISO/IEC 14443-4.NOTE For testing, the PCD shall transmit an I-Block with predetermined random data of 1kByte in the INF-field using an extended frame as defined in ISO/IEC 14443-3.

7.2.5.4 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.

"

Page 198 of ISO/IEC 10373-6:2011

Add following new Annexes at the end of Annex I:

"

Annex Amd.5 A
(normative)

IQ segment, noise and ISI analysis tool

This annex give MATLAB-style code that evaluates the data as captured by an oscilloscope during test procedure 7.1.6.2.

Assumptions:

The PSK order M (number of constellation points) is known, together with the extension of the IQ segment (the portion of the circle used for modulation), i.e. the phase difference between the extreme 2 constellation points. Moreover, the random data within the INF-Field is *a priori* known by the evaluation algorithm and is denoted as reference data.

Algorithm initialization:

1. Read scope data from .csv file (voltage versus time)
2. Calculate sample time (=1/fs) based on time values :
   tsample = (time[Nsamples-1]-t_start)/(Nsamples-1);
3. Software “resample” to pre-defined rate such that
   Nsamples_per_symbol = (1/tsample)/symrate
   becomes certain pre-defined integer (minimum 8 samples per carrier period).

Main Algorithm:

4. Run the algorithm that evaluates the *interference* (both on the IQ plane signal and on the phase signal) and the *phase noise* based on a few basic steps:
   - Align sample grid to symbols. This is obtained by demodulating the signal into the IQ plane, using different symbol grids, and selecting the grid which maximizes the variance of the demodulated signal.
   - Demodulate by windowed DFT per symbol. The DFT is operated over a symbol time and evaluated at the carrier frequency. Prior to DFT a windowing by a time domain mask is done, that only selects the ending portion of a symbol. The value of this DFT at the carrier frequency equals the integral of the baseband signal during the windowing time.
   - Align symbols, extract the transmitted frame. This is done by searching the maximum correlation between the reference symbols with the demodulated ones.
   - Evaluate ISI and noise. This is achieved with a system identification approach using a Linear Least Squares (LLS) fit.

About the last point the following sub-steps are taken:

- The received frame is fitted in a linear-least-squares sense to an arbitrary linear model on IQ plane.
- ISI and detuning are deduced univocally and robustly against noise from this model.
- A noiseless reference is built based on this model. Noise is estimated as a difference between this reference and the received signal.
- The RMS value of the noise in the phase direction (phase noise) is reported after a differentiating filter which removes the low frequency components. This D-RMS value is used to decide if the tests passes or fails.

```
%% ASSUME knowledge of:
%%   fs, fSymb, fc
%%   ASSUME: fs is integer multiple of fc
%%   M       (PSK modulation order)
%%   DphiDeg (IQ-segment sector, DEGREES)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% fc                    =   carrier frequency
% fs                    =   sample frequency, integer multiple of fc
% fSymb                 =   symbol frequency (=1/etu)
% M                     =   PSK Modulation order
% DphiDeg               =   IQ-segment extension (degrees)
% Input_fs,NN           =   scope data resampled at fs
% Symbols_TX_REF, nTX   =   TX symbols (a-priory known for SyncFrame 4-8-16 PSK)
function DFTsymbs = Eval_PCD_2_PICC(fc, fs, fSymb, M, DphiDeg, Input_fs, NN, Symbols_TX_REF, nTX)

PN_MAX_RMS_SI = 0.033;
ISIm_SI_Max1  = 1.6;
ISIm_SI_Max2  = 0.5;
ISId_DEG_MAX  = 20;
rad2deg       = 180/pi;

%Modulation Parameter definition based on inputs
carrPerSymb         = fc/fSymb;
samplesPerSymb      = fs/fSymb;
SymbIntDEG          = DphiDeg/(M-1);
Nc                  = fc/fSymb;


%Windowing definition
%% Window can use last 1 carrier in 1 symbol
smplHC          = floor(samplesPerSymb/carrPerSymb/2);
windowHC        = [zeros(1,samplesPerSymb-smplHC),2*carrPerSymb*ones(1,smplHC)];%for symbol grid
alignment
smpl1C          = floor(samplesPerSymb/carrPerSymb);
window1C        = [zeros(1,samplesPerSymb-smpl1C),carrPerSymb*ones(1,smpl1C)];  %Default for
Demodulation

fprintf('\n Analyzing input...\n');

%% ######################
%% (1) Demodulate signal
%% ######################
[DFTsymbs, nSymbDFT, IQ_seg_meas_rad, optGridPos] = demodulateCarrierDFT(Input_fs, NN,
samplesPerSymb, windowHC, window1C, carrPerSymb);


%% ######################
%% (2) Get Reference signal
%% ######################
    Symbols_FRM_RX = extractOneFrame(DFTsymbs,nSymbDFT,Symbols_TX_REF, nTX);


%% ######################
%% (3) Estimate params
%% ######################
Ntaps = 4;
[hVec, RMSerr]= LLS_Cplx_Channel_Fit_NTaps([1,1,1,1,Symbols_TX_REF],[1,1,1,1,Symbols_FRM_RX], Ntaps);
h0 = hVec(1); h1=hVec(2);

%% ###########################################################################

%% Estimate main ISI parameters
ISIratio               = abs(h1)./abs(h0); %ISIangle=asin(L/R), where L=distance 2 outer pts in 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(DphiDeg/2/180*pi)); %size of ISI cloud in Degrees
ISIangleDEG            = ISIangle*rad2deg;
ISIm_SI                = ISIangleDEG/SymbIntDEG;
detuningAngleDEG       = (angle(h1)-angle(h0))*rad2deg;
DTF = detuningAngleDEG/(360*Nc);

fprintf('------------\n');
fprintf('ISI Measures\n');
fprintf('------------\n');
fprintf('IQ segment (measured)  = %2.2f DEG \n',IQ_seg_meas_rad*rad2deg);
fprintf('Symbol Interval        = %2.2f DEG \n',SymbIntDEG);
fprintf('ISI magnitude          = %05.2f [DEG]  = %2.2f [Symb.Int]\n',ISIangleDEG,ISIm_SI);
```

```
fprintf('ISI rotation           = %05.2f [DEG]\n',detuningAngleDEG);

%% Estimate phase noise
%reconstruct linear noiseless signal @ Channel output
trailSilence = 10;
Symbols_Chout_k_REF = filter(hVec,1,[ones(1,trailSilence),Symbols_TX_REF]);
Symbols_Chout_k_REF = Symbols_Chout_k_REF(1+trailSilence:end);



%Measure phase noise based on this REF
phaseNoise         = angle(Symbols_FRM_RX)-angle(Symbols_Chout_k_REF);
%Condition Phase noise (differential)
phaseNoise         = diff(phaseNoise); %removes Low Freq PN and any residual DC
dphaseNoiseP2P     = max(phaseNoise)-min(phaseNoise);
dphaseNoiseP2PDEG  = dphaseNoiseP2P*rad2deg;
dphaseNoiseP2P_SI  = dphaseNoiseP2PDEG/SymbIntDEG;
dphaseNoiseRMS_DEG = sqrt(mean(phaseNoise.^2))*rad2deg;
dphaseNoiseRMS_SI  = dphaseNoiseRMS_DEG/SymbIntDEG;
%Any non linearity and wrong constellation point definition fall into this measure
fprintf('--------------\n');
fprintf('Noise Measures\n');
fprintf('--------------\n');
fprintf('D-Phase noise RMS      = %05.2f [DEG]  = %2.3f
[Symb.Int]\n',dphaseNoiseRMS_DEG,dphaseNoiseRMS_SI);
fprintf('D-Phase noise P2P      = %05.2f [DEG]  = %2.3f
[Symb.Int]\n',dphaseNoiseP2PDEG,dphaseNoiseP2P_SI);

%% Test Result
%% ############
if (abs(detuningAngleDEG) < ISId_DEG_MAX)
    passISI = (ISIm_SI < ISIm_SI_Max1);
else
    passISI = (ISIm_SI < ISIm_SI_Max2);
end
passNoise = dphaseNoiseRMS_SI < PN_MAX_RMS_SI;

if(~passNoise)
    fprintf('\n Test failed, due to too large noise \n');
end
if(~passISI)
    fprintf('\n Test failed, due to too large Interference \n');
end
if(passNoise & passISI)
    fprintf('\n Test Passed \n');
end
```

```
% Input_fs,NN      =   scope data resampled at fs
% windowAlign      =   narrow window used for symbol grid alignment (e.g. boxcar 1/2 carrier long)
% windowDemod      =   window used for symbol demodulation (e.g. boxcar 2 carriers long)
%
%OUTPUTS:
% IQ_seg_meas_rad  =   measured IQ segment (always slightly smaller than actual one due to ISI).
% optGridPos       =   sample offset denoting where a symbol starts

function  [DFTsymbs, nSymbDFT, IQ_seg_meas_rad, optGridPos] = ...
          demodulateCarrierDFT(Input_fs, NN, samplesPerSymb, windowAlign, windowDemod, carrPerSymb)

NumSymbs            = floor(NN/samplesPerSymb);

fprintf('Searching Symbol Grid...\n');
%% ######################
%% (1) Find Symbol Grid
%% ######################
%1a- First Demodulation, with random phase, Find Begin of preamble, where to start symbol grid search
gridPos = 0;
nSymbDFT     = NumSymbs-1;        %skip last symbol
DFTsymbs     = zeros(1,nSymbDFT); %allocate DFT output
for iSymbol = 0:nSymbDFT-1
    symbolStart           = gridPos + iSymbol*samplesPerSymb;
    DFTsymbs(iSymbol+1)   = windowed_DFT_1val(Input_fs, NN, ...
                                          symbolStart, samplesPerSymb, windowAlign, carrPerSymb);
end
noSilenceSymbIdx = find(180/pi*diff(angle(DFTsymbs)) > 1);
```

```
symbolBegin   = noSilenceSymbIdx(1);

%1b- Now search symbol grid on the beginning of preamble
nSymbDFTShort = 64;
DFTsymbs     = zeros(1,nSymbDFT); %allocate DFT output
varphiMax    = -1;
for gridPos=0:samplesPerSymb-1
    for iSymbol = symbolBegin:symbolBegin+nSymbDFTShort
        symbolStart           = gridPos + iSymbol*samplesPerSymb;
        DFTsymbs(iSymbol+1)   = windowed_DFT_1val(Input_fs, NN, ...
                                    symbolStart, samplesPerSymb, windowAlign, carrPerSymb);
    end
    varDFT                    =
complexVariance(DFTsymbs(symbolBegin:symbolBegin+nSymbDFTShort),nSymbDFTShort+1);
    if (varDFT > varphiMax) % new maximum found
        varphiMax = varDFT; optGridPos = gridPos;
    end
end
optGridPos = max(optGridPos-1,0); %safer 1 sample earlier than later


fprintf('Operating Windowed DFT Demodulation...\n');
%% ############################
%% (2) Calc DFT of each symbol (demodulate to cplx baseband)
%% ############################
for iSymbol = 0:nSymbDFT-1
    symbolStart         = optGridPos + iSymbol*samplesPerSymb;
    DFTsymbs(iSymbol+1) = windowed_DFT_1val(Input_fs, NN, ...
                                        symbolStart, samplesPerSymb, windowDemod, carrPerSymb);
end
DFTsymbs        = ReAlignSilencePhase(DFTsymbs, nSymbDFT, 0);
IQ_seg_meas_rad = max(angle(DFTsymbs))-min(angle(DFTsymbs)); %Measure IQ segment (approx)
```

```
function Symbols_FRM_RX = extractOneFrame(DFTsymbs,nSymbDFT,Symbols_TX_REF, nTX)
%correlation based symbol alignment
phasesDemod       = angle(DFTsymbs);
preambleSYF_phase = angle(Symbols_TX_REF);       %tx phasesDemod

autoCorrVal         = sum(preambleSYF_phase.^2);
minCrossCorr        = autoCorrVal/2;
maxCorr             = -1;
idxStart            = -1;

for iSymbol = 0:nSymbDFT-1-nTX  %loop over rx symbols (longer than SYF)
    corrOut = 0;
    for idxSYF = 0:nTX-1        %loop over preamble phasesDemod
        corrOut = corrOut + preambleSYF_phase(1+idxSYF)*phasesDemod(1+iSymbol+idxSYF);
    end
    if corrOut > maxCorr
        maxCorr = corrOut;
        idxStart= iSymbol;
    end
end
if (maxCorr < minCrossCorr)
    fprintf('\n Symbol Alignment Failed, wrong inputs ?');
end
Symbols_FRM_RX = DFTsymbs(1+idxStart:1+idxStart+nTX-1);
```

```
function symbolDecisions = sliceSymbols2PSK(Symbols_RX)

minAngle = min(angle(Symbols_RX));  %level 0
maxAngle = max(angle(Symbols_RX));  %level 1
ThrAngle = (minAngle+maxAngle)/2;

N1 = length(Symbols_RX);
symbolDecisions = zeros(1,N1);
for jj=1:N1
    if angle(Symbols_RX(jj))>ThrAngle
        symbolDecisions(jj) = exp(i*maxAngle);
    else
        symbolDecisions(jj) = exp(i*minAngle);
    end
end
```

```
%inputVec, vecSize   : The measurement data
%startPos, Nsmpls    : Defines the portion of data for current DFT
```

```
%window              : Weights the selected portion of data
%freqIdx             : 0..Nsmpls-1;    freqIdx = freq/fs*Nsmpls, is a normalized frequency
%                    : At the carrier: freqIdx = fc/fs*fs/fSymb = fc/fSymb = carriers/symbol
function  DFTval = windowed_DFT_1val(inputVec, vecSize, ...
                                     startPos, Nsmpls, window, ...
                                     freqIdx)
if (startPos+Nsmpls > vecSize)
    fprintf('\n DFTval Error! Out of boundary! \n');
    DFTval = [];
    return;
end

accVal = 0;
for jj=startPos:startPos+Nsmpls-1
    iSmpl  = jj-startPos;
    accVal = accVal + inputVec(1+jj)*window(1+iSmpl)*exp(-i*2*pi*freqIdx*iSmpl/Nsmpls);
end

DFTval = 2*accVal/Nsmpls;
```

```
%var(X) = E{(X-mu)Conjugate(X-mu)}
function var = complexVariance(cplxVecIn, vecLen)

vecLen = length(cplxVecIn); %Remove in C

if (vecLen ==1)
    var = 0;
else
    mu  = sum(cplxVecIn)/vecLen;
    var = sum((cplxVecIn-mu).*conj(cplxVecIn-mu))/(vecLen-1);
end
```

```
function cplxVecOut = ReAlignSilencePhase(cplxVecIn, vecLen, silencePhaseRef)

phaseInDeg =180/pi*atan2(imag(cplxVecIn),real(cplxVecIn));

%find begin of silence
phaseDegNoSilence = 10;
phaseDiffDegAbs   = phaseDegNoSilence;
idxBegin = 1;
while (phaseDiffDegAbs>=phaseDegNoSilence && idxBegin < vecLen)
    phaseDiffDegAbs = abs(phaseInDeg(1+idxBegin)-phaseInDeg(1+idxBegin-1));
    idxBegin = idxBegin+1;
end

%find end of silence and avg phase of unmod carrier
phaseDiffDegAbs = 0;
idxEnd = idxBegin;
avgUnModPhase = 0;
count    = 0;
while (phaseDiffDegAbs<phaseDegNoSilence && idxEnd < vecLen)
    phaseDiffDegAbs = abs(phaseInDeg(1+idxEnd)-phaseInDeg(1+idxEnd-1));
    avgUnModPhase = avgUnModPhase + phaseInDeg(1+idxEnd-1);
    count    = count+1;
    idxEnd   = idxEnd+1;
end
avgUnModPhase = avgUnModPhase/count/180*pi;

%rotate to Reference phase of silence
cplxVecOut  = cplxVecIn.*exp(-i*(avgUnModPhase-silencePhaseRef));
```

```
function [hVec, RMSerr] = LLS_Cplx_Channel_Fit_NTaps(Symbols_TX, Symbols_RX, Ntaps)
%    [ 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, a-priori, noiseless      TX
%y = output, observation, noisy     RX

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

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

%prepare data
m       = length(Symbols_TX)-Ntaps+1;  %Column size ~ observation Length
X       = zeros(m,Ntaps);
for iCol = 0:Ntaps-1
    X(:,iCol+1) = Symbols_TX(end-iCol  : -1 : Ntaps-iCol)'; %(m x Ntaps)  (a-priori, noiseless)
end
y       = Symbols_RX(end:-1:Ntaps)';                         %(m x 1)      (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 = hVec';

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

Annex Amd.5 B
(informative)

Example Test report

This annex shows example test reports as could be produced by the IQ segment, noise and ISI analysis tool for the data rate 3*fc*/4.

```
Example1: Pass
------------
ISI Measures
------------
IQ segment (measured)  = 55.26 DEG
Symbol Interval        = 8.00 DEG
ISI magnitude          = 11.68 [DEG]  = 1.46 [Symb.Int]
ISI rotation           = 02.40 [DEG]
--------------
Noise Measures
--------------
D-Phase noise RMS      = 00.14 [DEG]  = 0.017 [Symb.Int]
D-Phase noise P2P      = 00.75 [DEG]  = 0.094 [Symb.Int]

 Test Passed

Example2: Fail due to noise
------------
ISI Measures
------------
IQ segment (measured)  = 61.73 DEG
Symbol Interval        = 8.00 DEG
ISI magnitude          = 11.79 [DEG]  = 1.47 [Symb.Int]
ISI rotation           = 00.97 [DEG]
--------------
Noise Measures
--------------
D-Phase noise RMS      = 00.48 [DEG]  = 0.060 [Symb.Int]
D-Phase noise P2P      = 02.32 [DEG]  = 0.290 [Symb.Int]

 Test failed, due to too large noise

Example3: Fail due to ISI
------------
ISI Measures
------------
IQ segment (measured)  = 52.54 DEG
Symbol Interval        = 8.00 DEG
ISI magnitude          = 19.07 [DEG]  = 2.38 [Symb.Int]
ISI rotation           = 16.70 [DEG]
--------------
Noise Measures
--------------
D-Phase noise RMS      = 00.12 [DEG]  = 0.015 [Symb.Int]
D-Phase noise P2P      = 00.72 [DEG]  = 0.090 [Symb.Int]

 Test failed, due to too large Interference
```

Annex Amd.5 C
(informative)

PCD ISI conditioning for PICC reception test

To test PICC reception under worst ISI conditions (conditions 2 through 5 of section 7.2.7.2), we are interested in creating test signals with worst case inter-symbol interference. Rather than using a physical worst-case antenna resonator (requiring a dedicated matching network for each amount of inter-symbol interference that is to be generated), the inter-symbol interference is created digitally before entering the PCD antenna. The PCD antenna will introduce little extra inter-symbol interference itself by using the wide-band ISO matching network as described in A.2.2 of the ISO 10373-6 document. The digital inter-symbol interference creation allows flexible test signal conditioning.

The digital pre-processing of the symbols that are to be transmitted in the test signal is a *baseband* representation of a PCD antenna resonator that *would* give rise to the desired level of inter-symbol interference, including rotation of the clouds. This is illustrated in Figure Amd5 C.1 —, where the transfer function H_{RF} describes the physical antenna resonator (having two complex conjugated poles at a frequency close to 13.56MHz). The effect of this resonator can be mimicked by a filter with transfer function H_{bb} in front of the upconversion mixer. The transfer function H'_{RF} represents a *wide-band* antenna plus matching network, such as described in A.2.2 of ISO10373-6, minimally effecting inter-symbol interference.

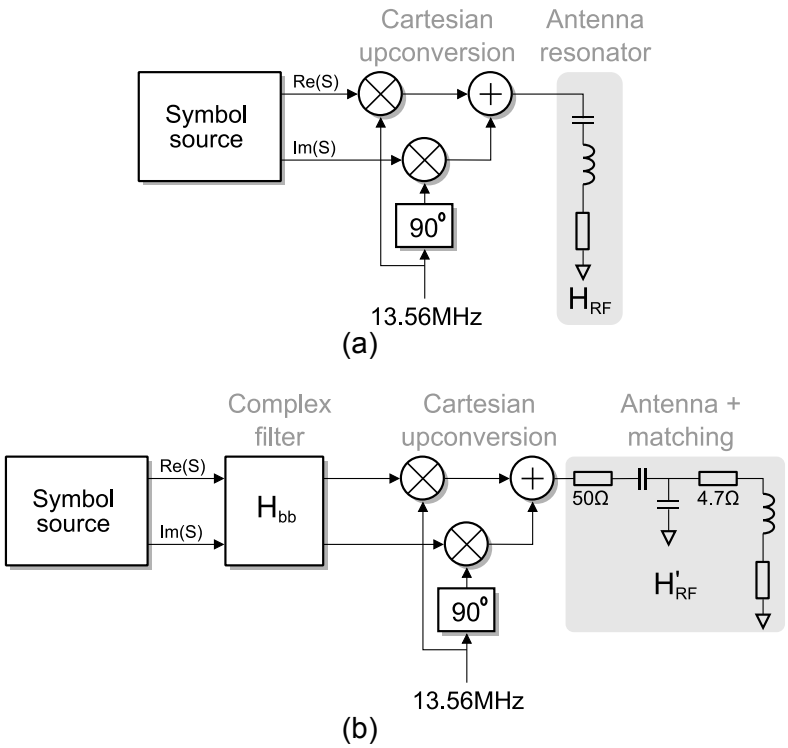

Figure Amd5 C.1 — Antenna resonator modeling using baseband complex filter. (a) Real PCD block schematic; (b) emulating H_{RF} using a complex filter H_{bb}.

A time-discrete baseband filter that creates the desired amount of ISI, including rotation, is described in the z-domain by:

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

with *p* the complex pole causing the ISI. If the sample rate f_{sr} of this filter is a multiple to the symbol rate f_{symb}, the pole location that yields the desired ISIm and ISId (magnitude and rotation) can be shown to be at:

$$p = (½ \cdot \sin(ISIm \cdot \Phi_{SI}) \cdot \exp(j\ ISId) / \sin(½\ \Phi_{seg}))^{(f_{symb}/f_{sr})}$$

with *j* the imaginary unit and Φ_{seg} the IQ segment used for PSK modulation.

Annex Amd.5 D
(informative)

PCD signal conditioning tool

This annex contains informative pseudo-code, describing how the five test conditions in the digital base-band domain could be created digitally.
The functions Condition1() through Condition5() assume an input array `Sin` of symbols that are already mapped to their PSK constellation points. This array therefore consists of complex numbers. For example, the rate 3fc/4 constellation point at +32 degrees is described by a complex number S, such that:

atan2(imag(S)/real(S)) == 32·π/180 rad.

The amplitude R of the constellation points, given by:

sqrt(imag(S)·imag(S) + real(S)·real(S)) == R

should be equal for all elements of `Sin`, such that all constellation points are located on a circle with radius R.

```
#include <complex>  // Standard Template Lib

// max phase noise:
#define PNRMSMAX          0.032 /* TBD */

// max ISI conditions 2 and 3
#define ISImmax1          1.5   /* TBD */
#define ISIdmax1          21.0  /* TBD */

// max ISI conditions 4 and 5
#define ISImmax2          0.52  /* TBD */
#define ISId2             60.0  /* TBD */

using namespace std;

// Definition of type cmplx:
typedef complex<double> cmplx;

////////////////////////////////
// Some helper functions: ////////
////////////////////////////////

double randn(void)
{ /* returns random number with normal distribution
     mean = 0; sigma = 1.0 */
  int i;
  double res = 0;
  for (i=0;i<4*12;++i)
    res += rand();
  return res/RAND_MAX/2.0 - 12.0;
}

cmplx ISIpole(double ISIm, double ISId, double IQseg, double M)
{ /* Calculates position of filter pole
     --- in ---
     ISIm:   ISI magnitude in units of symbol interval
     ISId:   ISI rotation in degrees
     IQseg:  IQ segment in degrees
     M:      PSK order
     --- out ---
     returns complex number representing position of pole */
  double  preal, pimag;
  double  deg2rad = atan(1)/45;  // conversion factor
  double  symint  = IQseg*deg2rad/(M-1);

  preal = sin(ISIm*symint)/sin(IQseg*deg2rad/2) *
          cos(ISId*deg2rad)/2.0;
```

```
48.   pimag = sin(ISIm*symint)/sin(IQseg*deg2rad/2) *
49.            sin(ISId*deg2rad)/2.0;
50.   return cmplx(preal,pimag);
51. }
52.
53. void ISIfilter(cmplx* Sin, int len, cmplx p, cmplx* Sout)
54. { /* Applies ISI based on complex pole
55.      --- in ---
56.      Sin:    clean input symbols
57.      len:    length of input symbol array
58.      p:      complex filter pole
59.      --- out ---
60.      Sout:   output symbols of function */
61.   int     i;
62.   double  amp;
63.
64.   amp = abs(Sin[0]); // amplitude R
65.   Sout[0] = p*amp + (1.0-p)*Sin[0]; // initial condition from silence
66.   for (i=1; i<len; i++)
67.     Sout[i] = p*Sout[i-1] + (1.0-p)*Sin[i];
68. }
69.
70. ////////////////////////////////
71. // The ConditionX functions: /////
72. ////////////////////////////////
73.
74. void Condition1(cmplx* Sin, int len, double IQseg, int M, cmplx* Sout)
75. { /* Applies condition 1 to Sin
76.      --- in ---
77.      Sin:    clean input symbols
78.      len:    length of input symbol array
79.      IQseg:  IQ segment in degrees
80.      M:      PSK order
81.      --- out ---
82.      Sout:   output symbols of function */
83.   int i;
84.   double symint, noiseang;
85.   cmplx noise;
86.
87.   symint = IQseg*atan(1)/45/(M-1);  // symbol interval
88.
89.   for (i=0; i<len; ++i) {
90.     noiseang = randn() * PNRMSMAX * symint;
91.       // randn(): gaussian noise
92.     noise    = polar(1.0, noiseang);  // exp(j*noiseang)
93.     /* noise contains complex noise product */
94.
95.     Sout[i] = Sin[i]*noise;  /* complex multiplication */
96.   }
97. }
98.
99. void Condition2(cmplx* Sin, int len, double IQseg, int M, cmplx* Sout)
100. { /* Applies condition 2 to Sin
101.      --- in ---
102.      Sin:     clean input symbols
103.      len:     length of input symbol array
104.      M:       number of PSK constellation points
105.      IQseg:   IQ segment in radians
106.      --- out ---
107.      Sout:    output symbols of function */
108.   cmplx   p;   // baseband time-discrete pole causing required ISI
109.
110.   p = ISIpole(ISImmax1, ISIdmax1, IQseg, M);
111.   ISIfilter(Sin,len, p, Sout);
112. }
113.
114. void Condition3(cmplx* Sin, int len, double IQseg, int M, cmplx* Sout)
115. { /* Applies condition 3 to Sin
116.      --- in ---
117.      Sin:     clean input symbols
118.      len:     length of input symbol array
119.      M:       number of PSK constellation points
120.      IQseg:   IQ segment in radians
121.      --- out ---
122.      Sout:    output symbols of function */
123.   cmplx   p;   // baseband time-discrete pole causing required ISI
```

```
124.
125.   p = ISIpole(ISImmax1, -ISIdmax1, IQseg, M);
126.   ISIfilter(Sin,len, p, Sout);
127. }
128.
129. void Condition4(cmplx* Sin, int len, double IQseg, int M, cmplx* Sout)
130. { /* Applies condition 4 to Sin
131.      --- in ---
132.      Sin:     clean input symbols
133.      len:     length of input symbol array
134.      M:       number of PSK constellation points
135.      IQseg:   IQ segment in radians
136.      --- out ---
137.      Sout:    output symbols of function */
138.   cmplx   p;   // baseband time-discrete pole causing required ISI
139.
140.   p = ISIpole(ISImmax2, ISId2, IQseg, M);
141.   ISIfilter(Sin,len, p, Sout);
142. }
143.
144. void Condition5(cmplx* Sin, int len, double IQseg, int M, cmplx* Sout)
145. { /* Applies condition 5 to Sin
146.      --- in ---
147.      Sin:     clean input symbols
148.      len:     length of input symbol array
149.      M:       number of PSK constellation points
150.      IQseg:   IQ segment in radians
151.      --- out ---
152.      Sout:    output symbols of function */
153.   cmplx   p;   // baseband time-discrete pole causing required ISI
154.
155.   p = ISIpole(ISImmax2, -ISId2, IQseg, M);
156.   ISIfilter(Sin,len, p, Sout);
157. }
```

"