भारतीय मानक **IS/ISO/IEC 10373 (Part 6) : 2020**

Indian Standard

व्यक्तिगत पहचान के लिए कार्ड और सुरक्षा उपकरण — परीक्षण पद्धतियाँ

भाग 6 संपर्क रहित प्रॉक्सिमिटी ऑब्जेक्ट्स

(पहला पुनरीक्षण)

Cards and Security Devices for Personal Identification — Methods of Test

Part 6 Contactless Proximity Objects

(First Revision)

ICS 35.240.15

भारतीय मानक ब्यूरो
BUREAU OF INDIAN STANDARDS
मानक भवन, 9 बहादुर शाह ज़फर मार्ग, नई दिल्ली - 110002
MANAK BHAVAN, 9 BAHADUR SHAH ZAFAR MARG
NEW DELHI - 110002
www.bis.gov.in www.standardsbis.in

September 2024 **₹ 5140**

NATIONAL FOREWORD

This Indian Standard (Part 6) (First Revision) which is identical to ISO/IEC 10373-6 : 2020 Cards and security devices for personal identification — Test methods — Part 6: 'Contactless proximity objects' issued by ISO 'International Organization Standardization' and IEC 'International Electrotechnical Commission' was adopted by the Bureau of Indian Standards on the recommendation of the Identification & Data capture techniques, Cards and Security Devices Sectional Committee and after approval of the Electronics and Information Technology Division Council.

This standard (Part 6) was first published in 2018 which was identical to ISO/IEC 10373-6 : 2016. This revision aligns this Indian Standard with ISO/IEC 10373 (Part 6) : 2020.

The main changes compared to the previous edition are as follows:

a) Enhancement of test methods for PCD load modulation reception and PICC transmission including introduction of active reference PICC and PICC amplitude and phase drift analysis tool;

b) Introduction of PICC Type A frame delay time (FDT) determination method; and

c) Extension of frame with error correction test methods.

This Indian Standard is published in several parts. The other parts in this series are:

Part 1 General characteristic tests
Part 2 Cards with magnetic stripes
Part 3 Integrated circuit cards with contacts and related interface devices
Part 5 Optical memory cards
Part 7 Contactless vicinity objects
Part 8 USB-ICC
Part 9 Optical memory cards — Holographic recording method

The text of IEC standard has been approved as suitable for publication as an Indian Standard without deviations. Certain conventions are, however, not identical to those used in Indian Standards. Attention is particularly drawn to the following:

a) Wherever the words 'International Standard' appears referring to this standard, they should be read as 'Indian Standard'; and

b) Comma (,) has been used as a decimal marker while in Indian Standards, the current practice is to use a point (.) as the decimal marker.

In this adopted standard, reference appears to certain International Standards for which Indian Standards also exist. The corresponding Indian Standards, which are to be substituted in their places, are listed below along with their degree of equivalence for editions indicated:

International Standard	*Corresponding Indian Standard*	*Degree of Equivalence*
ISO/IEC 7810, Identification cards — Physical characteristics	IS 14172 : 2024/ISO/IEC 7810 : 2019 + Amd 1 : 2024 Identification cards physical characteristics (*second revision*)	Identical
ISO/IEC 14443-1 : 2018 Cards and security devices for personal identification — Contactless proximity objects — Part 1: Physical characteristics	IS/ISO/IEC 14443-1 : 2018 Cards and security devices for personal identification Contactless proximity objects: Part 1 Physical characteristics	Identical

(*Continued on third cover*)

Contents

Page

Indian Standard

CARDS AND SECURITY DEVICES FOR PERSONAL IDENTIFICATION — METHODS OF TEST

PART 6 CONTACTLESS PROXIMITY OBJECTS

(First Revision)

1 Scope

The ISO/IEC 10373 series defines test methods for characteristics of identification cards according to the definition given in ISO/IEC 7810. Each test method is cross-referenced to one or more base standards, which can be ISO/IEC 7810 or one or more of the supplementary standards that define the information storage technologies employed in identification card applications.

NOTE 1 Criteria for acceptability do not form part of the ISO/IEC 10373 series, but can be found in the International Standards mentioned above.

This document defines test methods which are specific to proximity cards and objects, proximity coupling devices and proximity extended devices, defined in ISO/IEC 14443-1, ISO/IEC 14443-2, ISO/IEC 14443-3 and ISO/IEC 14443-4.

NOTE 2 Test methods defined in this document are intended to be performed separately. A given proximity card or object, proximity coupling device or proximity extended device, is not required to pass through all the tests sequentially.

ISO/IEC 10373-1 defines test methods which are common to one or more integrated circuit card technologies and other parts in the ISO/IEC 10373 series deal with other technology-specific tests.

The conformance test plan defined in Annex O specifies the list of tests applicable for each part of the ISO/IEC 14443 series.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 7810, *Identification cards — Physical characteristics*

ISO/IEC 14443-1:2018, *Cards and security devices for personal identification — Contactless proximity objects — Part 1: Physical characteristics*

ISO/IEC 14443-2:2020, *Cards and security devices for personal identification — Contactless proximity objects — Part 2: Radio frequency power and signal interface*

ISO/IEC 14443-3:2018, *Cards and security devices for personal identification — Contactless proximity objects — Part 3: Initialization and anticollision*

ISO/IEC 14443-4:2018, *Cards and security devices for personal identification — Contactless proximity objects — Part 4: Transmission protocol*

3 Terms, definitions, symbols and abbreviated terms

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 14443-1, ISO/IEC 14443-2, ISO/IEC 14443-3, ISO/IEC 14443-4 and the following apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at http://www.electropedia.org/

3.1.1
base standard
standard to which the *test method* (3.1.8) is used to verify conformance

3.1.2
CascadeLevels
number of cascade levels of the PICC

3.1.3
Command Set
set describing the PICC commands during initialization and anticollision

Note 1 to entry: See ISO/IEC 14443-3:2018, 6.4 for PICC Type A and ISO/IEC 14443-3:2018, 7.5 for PICC Type B.

3.1.4
loading effect
change in PCD antenna current caused by the presence of PICC(s) in the field due to the mutual coupling modifying the PCD antenna resonance and quality factor

3.1.5
Mute
no response within a specified timeout

EXAMPLE Expiration of FWT.

3.1.6
scenario
defined typical protocol and application specific communication to be used with the *test methods* (3.1.8) defined in this document

3.1.7
Test Initial State
TIS
element from PICC states that is the PICC state before performing a specific PICC command from *Command Set* (3.1.3)

3.1.8
test method
method for testing characteristics of identification cards for the purpose of confirming their compliance with International Standards

3.1.9
Test Target State
TTS
element from PICC states that is the PICC state after performing a specific PICC command from *Command Set* (3.1.3)

3.2 Symbols and abbreviated terms

For the purposes of this document, the symbols and abbreviated terms given in ISO/IEC 14443-1, ISO/IEC 14443-2, ISO/IEC 14443-3, ISO/IEC 14443-4 and the following apply.

NOTE Elements in bold square brackets **[ ]** are optional definitions.

Answer to ATTRIB(cid)	Answer to ATTRIB, i.e. (MBLI = mbli, CID = cid, CRC_B), with mbli an arbitrary hex value (see ISO/IEC 14443-3:2018, 7.11)
ATTRIB(cid, fsdi)	Default ATTRIB command with PUPI from ATQB, CID = cid and Maximum Frame Size Code value = fsdi, i.e. ('1D' PUPI cid fsdi '01 00' CRC_B)
DUT	Device under test
$I(c)_n$**(**inf **[**,CID = cid**] [**,NAD = nad**] [**,~CRC**])**	ISO/IEC 14443-4 I-block with chaining bit c∈{1,0}, block number n∈{1,0} and information field inf. By default no CID and no NAD will be transmitted. If CID = cid∈{0...15} is specified, it will be transmitted as second parameter. If NAD = nad∈{0...'FF'} is specified, it will be transmitted as third parameter (or second parameter if no CID is transmitted). If the literal '~CRC' is not specified, a valid CRC corresponding to the communication signal interface of the PICC will be transmitted by default (i.e. CRC_A or CRC_B)
IUT	Implementation Under Test (ISO/IEC 9646); within the scope of this document, IUT represents the PCD under the test
LT	Lower Tester (ISO/IEC 9646), the PICC-emulation part of the PCD-test-apparatus
N/A	Not applicable
PPS(cid, dri, dsi)	Default PPS request with CID = cid, DRI = dri and DSI = dsi, i.e. ('D' + cid '11' dsi × 4 + dri CRC_A)
R(ACK **[**,CID = cid**] [**,~CRC**])**$_n$	ISO/IEC 14443-4 R(ACK) block with block number n. The definition of the optional CID and ~CRC symbols is as described in the $I(c)_n$ block above
R(NAK **[**,CID = cid**][**,~CRC**])**$_n$	ISO/IEC 14443-4 R(NAK) block with block number n. The definition of the optional CID and ~CRC symbols is as described in the $I(c)_n$ block above
RATS(cid, fsdi)	Default RATS command with CID = cid and FSDI value = fsdi i.e. ('E0' fsdi × 16 + cid CRC_A)
READY(I)	READY state in cascade level I, I ∈ {1, 2, 3}; e.g. READY(2) is a PICC cascade level 2
READY*(I)	READY* state in cascade level I, I ∈ {1, 2, 3}; e.g. READY*(2) is a PICC cascade level 2
REQB(N)	REQB command with N as defined in ISO/IEC 14443-3:2018, 7.7
S(WTX)(WTXM [,CID = cid][,~CRC])	ISO/IEC 14443-4 S(WTX) block with parameter WTXM. The definition of the optional CID and ~CRC symbols is as described in the $I(c)_n$ block above

S(DESELECT **[**,CID = cid**]** **[**,~CRC**]**)	ISO/IEC 14443-4 S(DESELECT) block. The definition of the optional CID and ~CRC symbols is as described in the $I(c)_n$ block above
SAK(cascade)	the SELECT(I) answer with the cascade bit (bit 3) set to (1)b
SAK(complete)	the SELECT(I) answer with the cascade bit (bit 3) set to (0)b
SEL(c)	Select code of level c (i.e. SEL(1) = '93', SEL(2) = '95', SEL(3) = '97')
SELECT(I)	SELECT command of cascade level I, i.e. SELECT(1) = ('93 70' $UIDTX_1$ BCC CRC_A) SELECT(2) = ('95 70' $UIDTX_2$ BCC CRC_A) SELECT(3) = ('97 70' $UIDTX_3$ BCC CRC_A)
SLOTMARKER(n)	Slot-MARKER command with slot number n, i.e. (16 × (n − 1) + 5 CRC_B)
TB-PDU	Transmission Block Protocol Data Unit, which consists of either I-block, R-block or S-block
TEST_COMMAND_SEQUENCE1	Sequence of commands used for several PICC tests. NOTE Its definition depends on applicative layer and represents a standard transaction of the application supported by the DUT. The applicant may also provide a specified set of commands.
TEST_COMMAND1(1)	Default test command consisting of one unchained I-block NOTE This command depends on the negotiated maximum frame size value of the PICC.
TEST_COMMAND1(n), n > 1	Default test command consisting of n chained I-blocks (PCD chaining) NOTE This command depends on the negotiated maximum frame size value of the PICC.
$TEST\_COMMAND1(n)_k$	INF field of k'th I-block chain of TEST_COMMAND1(n) NOTE This command depends on the negotiated maximum frame size value of the PICC.
TEST_COMMAND2(n), n > 1	Default test command which expects a response consisting of n chained I-blocks NOTE This command depends on the negotiated maximum frame size value of the PCD.
TEST_COMMAND3	Default test command consisting of one I-block which needs more than FWT time for execution
TEST_COMMAND4	Default test command which expects a response of one I-block complying with the PICC transmission minimum frame length required for the PICC transmission test
TEST_RESPONSE1(n)	INF field of the response to TEST_COMMAND1(n) NOTE This response is assumed to be always unchained.

TEST_RESPONSE2(n)	Response to TEST_COMMAND2(n) NOTE This response depends on the negotiated maximum frame size value of the PCD.
$TEST\_RESPONSE2(n)_k$	INF field of k'th I-block chain of TEST_RESPONSE2(n) NOTE This response depends on the negotiated maximum frame size value of the PCD.
TEST_RESPONSE3	Response I-block to TEST_COMMAND3 NOTE This response is always assumed to be unchained.
TEST_RESPONSE4	Response I-block to TEST_COMMAND4
TM-PDU	Test Management Protocol Data Unit (ISO/IEC 9646-1, PDU)
t_{START}	Start of PICC transmission
$UIDTX_I$	Transmitted UID 32-bit data at cascade level I (see Table 1)
UT	Upper Tester (ISO/IEC 9646), the master part of the PCD-test-apparatus
UT_APDU	Upper Tester Application Protocol Data Unit: a packet of data to be sent by the PCD to the LT through the RF interface
V_{load}	DC voltage measured at connector CON3 of the Reference PICC
WUPB(N)	WUPB command with N as defined in ISO/IEC 14443-3:2018, 7.7
~X	Bit sequence consisting of the inverted bits of bit sequence X or any other bit sequence different from X
X[[a...b]]	Bit subsequence of bit sequence X consisting of the bits between position a and b included. If a > b then the sequence is empty
X[[n]]	Bit at position n of bit sequence X. First bit is at position 1
X[n]	Byte at position n of bit sequence X. First byte is at position 1 (i.e. X[n] = X[[(n − 1) × 8 + 1...n × 8]])

Table 1 — Mapping from UID to UIDTX

Cascade level	Single UID PICC	Double UID PICC	Triple UID PICC
$UIDTX_1$	UID0 UID1 UID2 UID3	'88' UID0 UID1 UID2	'88' UID0 UID1 UID2
$UIDTX_2$	—	UID3 UID4 UID5 UID6	'88' UID3 UID4 UID5
$UIDTX_3$	—	—	UID6 UID7 UID8 UID9

4 Default items applicable to the test methods

4.1 Test environment

Unless otherwise specified, testing shall take place in an environment of temperature 23 °C ± 3 °C (73 °F ± 5 °F) and of relative humidity 25 % to 75 %.

4.2 Pre-conditioning

No environmental pre-conditioning of PICCs or PCDs is required by the test methods in this document.

4.3 Setup tolerances

The following absolute tolerances shall be used when adjusting the Test PCD assembly modulation waveform:

1) for timings (t_1, t_2, t_3, t_5, t_6, t_r, t_f):
   a) $\pm 1/f_c$ for a PCD to PICC bit rate of $f_c/128$
   b) $\pm 0{,}5/f_c$ for a PCD to PICC bit rate of $f_c/64$
   c) $\pm 0{,}3/f_c$ for PCD to PICC bit rates higher than $f_c/64$
2) for envelope overshoot, Type A, PCD to PICC bit rate of $f_c/128$: ±1 % of $H_{INITIAL}$
3) for envelope overshoot, Type A, PCD to PICC bit rates higher than $f_c/128$: ±0,01 × (1-a)
4) for envelope overshoot and undershoot, Type B: ±0,01 × (1-b)
5) for the modulation index m: ±0,5 %
6) for the pulse shape factor a: ±0,02
7) for PCD field envelope during 60 % of t_2: ±0,5 % of $H_{INITIAL}$

Unless otherwise specified, a tolerance of ±5 % shall be applied to the quantity values given to specify the characteristics of the test equipment (e.g. linear dimensions) and the test method procedures (e.g. test equipment adjustments).

4.4 Spurious inductance

Resistors and capacitors should have negligible inductance.

4.5 Measurement uncertainty

The measurement uncertainty for each quantity determined by these test methods shall be stated in the test report.

Basic information is given in ISO/IEC Guide 98-3.

4.6 DUT position

Unless otherwise specified, the PICC and Reference PICC antennas shall be centered on the sense coil a of the test PCD assembly.

4.7 Test conditions for PCD

Unless otherwise specified, the test conditions defined in Table 2 shall be applied.

Table 2 — Test conditions for PCD

Conditions	Values
Type	Type A and Type B
Test positions	Position 0: See Table 3 Position Z_{max}: See Table 3
Reference PICCs	Reference PICC 1, Reference PICC 2 and Reference PICC 3 In accordance with the support of optional PICC classes as declared by the PCD manufacturer in Table 3: — Reference PICC 4 if PICC Class 4 is supported, — Reference PICC 5 if PICC Class 5 is supported, — Reference PICC 6 if PICC Class 6 is supported.

The information defined in Table 3 shall be provided by the PCD manufacturer.

Table 3 — PCD manufacturer information

Parameter	Description	Unit
Position 0	Position and orientation of the Reference PICCs on the PCD surface. This position may be PICC classes dependent.	
Position Z_{max}	Position with maximum operating distance on the Z axis[a]. This position may be PICC classes dependent.	
Temperature range	Minimum and maximum temperature values.	°C
Optional PICC classes	List of supported optional PICC classes (4 and/or 5 and/or 6).	
PCD to PICC supported bit rates	List of supported optional PCD to PICC bit rates.	
PICC to PCD supported bit rates	List of supported optional PICC to PCD bit rates.	
Maximum frame size supported	Maximum frame size in reception.	bytes
PCD to PICC frame with error correction supported	Frame with error correction from PCD to PICC.	
PICC to PCD frame with error correction supported	Frame with error correction from PICC to PCD.	

[a] Z axis shall be perpendicular to the PCD surface through Position 0. If the PCD surface is not flat, Z axis shall correspond to the axis along which PICCs would habitually be held to the PCD and shall be coherent with PCD ergonomics; if not, the test laboratory may choose to redefine it (directionally).

Unless otherwise specified, the values defined in Table 4 shall be used to adjust PCD-test-apparatus parameters.

Table 4 — Values of the PCD-test-apparatus parameters unless otherwise specified

Parameter	Value	Applies to
PCD to PICC and PICC to PCD bit rates	$f_c/128$	Type A and Type B
Load modulation amplitude	More than 20 mV at H_{min}	Type A and Type B
Reference PICCs resonance frequency	16,5 MHz	Type A and Type B
J1 setting	position 'a'	Type A and Type B
J2 setting	position ‘a’	Type A and Type B
Reference PICCs position	Position Z_{max}	Type A and Type B
Start Of Frame (SOF) timing	10 etu "0" followed by 2 etu "1"	Type B
End Of Frame (EOF) timing	10 etu "0"	Type B
Extra Guard Time (EGT) timing	0 etu	Type B
TR0 for ATQB and DESELECT	$200/f_s$	Type B

Table 4 *(continued)*

Parameter	Value	Applies to
Frame waiting time	Any value as specified in ISO/IEC 14443-4:2018, 7.2	Type A and Type B
UID	Any of the size and contents as specified in ISO/IEC 14443-3:2018, 6.5.4	Type A
TR1	$140/f_s$	Type B
FSCI	8	Type A
Maximum Frame Size Code in ATQB	8	Type B

4.8 Test conditions for PICC

Unless otherwise specified, the test conditions defined in Table 5 shall be applied:

Table 5 — Test conditions for PICC

Conditions	Values	
Field strength[a]	For PICC Class 1:	1,5 A/m, 2,5 A/m, 4,5 A/m and 7,5 A/m
	For PICC Class 2 and Class 3:	1,5 A/m, 2,5 A/m, 4,5 A/m and 8,5 A/m
	For PICC Class 4:	2 A/m, 4 A/m, 7 A/m and 12 A/m
	For PICC Class 5:	2,5 A/m, 4,5 A/m, 8 A/m and 14 A/m
	For PICC Class 6:	4,5 A/m, 7 A/m, 11 A/m and 18 A/m

[a] Any additional field strength values between H_{min} and H_{max} may be applied.

The information defined in Table 6 shall be provided by the PICC manufacturer.

Table 6 — PICC manufacturer information

Parameter	Description	Unit
Location of the external rectangle/ circle of the claimed PICC class[a]	Drawing with dimensions of PICC outside shape and the position of the external rectangle/circle of the claimed PICC class.	
PICC class (optional)[a]	Claimed PICC class.	
Resonance frequency range (optional)	Minimum and maximum resonance frequency.	MHz
Communication signal interface	Supported communication signal interface(s): — Type A — Type B — Type A and Type B	
Temperature range	Minimum and maximum operating temperature	°C
PCD to PICC supported bit rates	List of supported optional PCD to PICC bit rates	
PICC to PCD supported bit rates	List of supported optional PICC to PCD bit rates	
Same bit rate for both directions	Indication if only same bit rate from PCD to PICC and from PICC to PCD is supported	
Random or fixed UID (Type A) or PUPI (Type B)	Indication whether the UID (Type A) or PUPI (Type B) is random or fixed	
Maximum frame size supported	Maximum frame size in reception	bytes

[a] If not provided, test methods for PICC Class 1 shall be used.

Table 6 *(continued)*

Parameter	Description	Unit
PCD to PICC frame with error correction supported	Frame with error correction from PCD to PICC	
PICC to PCD frame with error correction supported	Frame with error correction from PICC to PCD	
TEST_COMMAND_SEQUENCE1	See 0.2.1	
TEST_COMMAND1	See 0.2.1	
TEST_COMMAND2	See 0.2.1	
TEST_COMMAND3	See 0.2.1	
TEST_COMMAND4	See 0.2.1	
[a] If not provided, test methods for PICC Class 1 shall be used.		

Unless otherwise specified, the values defined in Table 7 shall be used to adjust PICC-test-apparatus parameters.

Table 7 — Values of the PICC-test-apparatus parameters unless otherwise specified

Parameter	Value	Applies to
Parameters applicable for all PCD to PICC bit rates		
FSDI	8	Type A
Start Of Frame (SOF) timing	10 etu "0" followed by 2 etu "1"	Type B
End Of Frame (EOF) timing	10 etu "0"	Type B
Extra Guard Time (EGT) timing	0 etu	Type B
Maximum Frame Size Code in ATTRIB	8	Type B
Parameters applicable for a PCD to PICC bit rate of $f_c/128$		
PCD field envelope during 60 % of t_2	0,5 %	Type A
t_1	$40/f_c$	Type A
t_2	$7/f_c$	Type A
t_3	$12/f_c$	Type A
t_4	$6/f_c$	Type A
Overshoot	0	Type A and Type B
Modulation index m	12 %	Type B
Rise time t_r, fall time t_f	$12/f_c$	Type B
Parameters applicable for a PCD to PICC bit rate of $f_c/64$		
a	0,1	Type A
t_1	$18/f_c$	Type A
t_5	$15/f_c$	Type A
t_6	$9/f_c$	Type A
Overshoot	0	Type A and Type B
Modulation index m	12 %	Type B
Rise time t_r, fall time t_f	$10/f_c$	Type B
Parameters applicable for a PCD to PICC bit rate of $f_c/32$		
a	0,2	Type A
t_1	$9/f_c$	Type A
t_5	$7/f_c$	Type A
t_6	$8/f_c$	Type A
Overshoot	0	Type A and Type B

Table 7 *(continued)*

Parameter	Value	Applies to
Modulation index m	12 %	Type B
Rise time t_r, fall time t_f	$8/f_c$	Type B
Parameters applicable for a PCD to PICC bit rate of $f_c/16$		
a	0,4	Type A
t_1	$5/f_c$	Type A
t_5	$4/f_c$	Type A
t_6	$5/f_c$	Type A
Overshoot	0	Type A and Type B
Modulation index m	12 %	Type B
Rise time t_r, fall time t_f	$6/f_c$	Type B
Parameters applicable for a PCD to PICC bit rate of $f_c/8$		
Overshoot	0	Type A and Type B
Modulation index m	8 % for short modulation pulses	Type A and Type B
Rise time t_r, fall time t_f	$5/f_c$	Type A and Type B
Parameters applicable for a PCD to PICC bit rate of $f_c/4$		
Overshoot	0	Type A and Type B
Modulation index m	8 % for short modulation pulses	Type A and Type B
Rise time t_r, fall time t_f	$4/f_c$	Type A and Type B
Parameters applicable for a PCD to PICC bit rate of $f_c/2$		
Overshoot	0	Type A and Type B
Modulation index m	8 % for short modulation pulses	Type A and Type B
Rise time t_r, fall time t_f	$3/f_c$	Type A and Type B
Parameters applicable for a PCD to PICC bit rate of $3f_c/4$ and $3f_c/2$		
PR	56°	Type A and Type B
ISI_d	0	Type A and Type B
ISI_m	1	Type A and Type B
Phase noise	0,03	Type A and Type B
Parameters applicable for a PCD to PICC bit rate of f_c and $2f_c$		
PR	60°	Type A and Type B
ISI_d	0	Type A and Type B
ISI_m	1	Type A and Type B
Phase noise	0,0125	Type A and Type B

5 Apparatus and circuits for test of ISO/IEC 14443-1 and ISO/IEC 14443-2 parameters

5.1 Overview

This clause defines the test apparatus and test circuits for verifying the operation of a PICC or a PCD according to ISO/IEC 14443-1 and ISO/IEC 14443-2. The test apparatus includes the following:

1) measurement instruments (see 5.2);

2) calibration coil (see 5.3);

3) Test PCD assembly (see 5.4);

4) Reference PICC and Active Reference PICC (see 5.5);

5) PICC transmission test setup (see 5.6);

6) EMD test setup (see 5.7).

These are described in the following subclauses.

5.2 Minimum requirements for measurement instruments — Oscilloscope

The digital sampling oscilloscope shall be capable of sampling at a rate of at least 500 million samples per second with a resolution of at least 8 bits at optimum scaling and shall have an overall minimum bandwidth of 250 MHz. The oscilloscope should have the capability to output the sampled data as a text file to facilitate mathematical and other operations such as windowing on the sampled data using software programs (see Annex E and Annex F).

NOTE The overall bandwidth is the combination of oscilloscope and probing system bandwidth.

5.3 Calibration coils

5.3.1 General

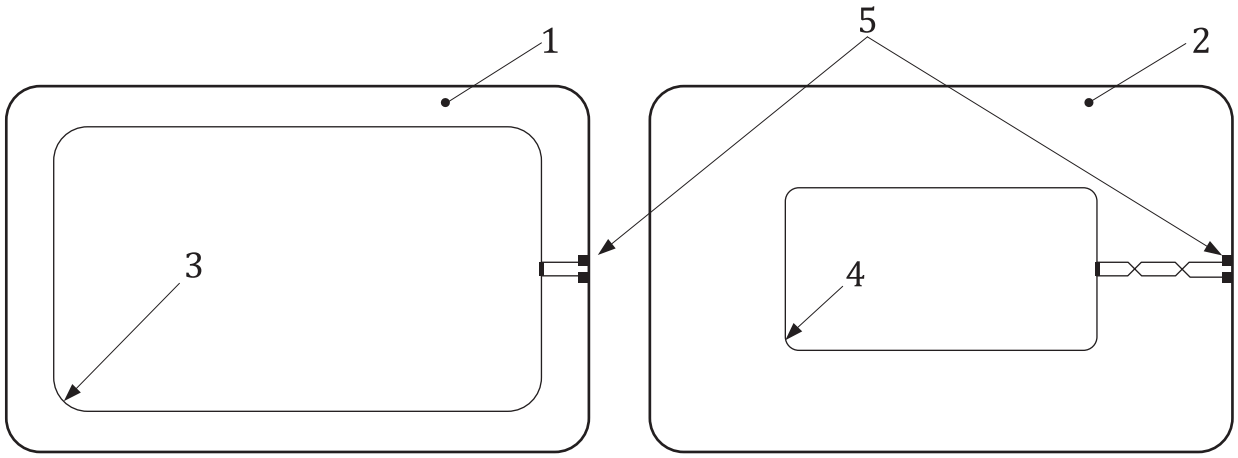
This subclause defines the size, thickness and characteristics of the calibration coils 1 and 2.

Calibration coil 1 shall be used only in Test PCD assembly 1 and calibration coil 2 shall be used only in Test PCD assembly 2.

5.3.2 Size of the calibration coil card

The calibration coil card shall consist of an area which has the height and width of an ID-1 type defined in ISO/IEC 7810 containing a single turn coil concentric with the card outline (see Figure 1).

Dimensions in millimetres



Key

1 calibration coil 1 (ISO/IEC 7810 ID-1 outline)
2 calibration coil 2 (ISO/IEC 7810 ID-1 outline)
3 coil 72 × 42, 1 turn
4 coil 46 × 24, 1 turn
5 connections

NOTE Drawings are not to scale.

Figure 1 — Calibration coils 1 and 2

5.3.3 Thickness and material of the calibration coil card

The thickness of the calibration coil card shall be less than that of an ID-1 card. It shall be constructed of a suitable insulating material.

5.3.4 Coil characteristics

The coil on the calibration coil card shall have one turn. Relative dimensional tolerance shall be ±2 %.

The outer size of the calibration coil 1 shall be 72 mm × 42 mm with corner radius 5 mm.

NOTE 1 The area over which the field is integrated is approximately 3 000 mm^2.

NOTE 2 At 13,56 MHz the approximate inductance is 250 nH and the approximate resistance is 0,4 Ω.

The open circuit calibration factor for the calibration coil 1 is 0,318 V (rms) per A/m (rms) [equivalent to 900 mV (peak-to-peak) per A/m (rms)].

The outer size of the calibration coil 2 shall be 46 mm × 24 mm with corner radius 2 mm.

NOTE 3 The area over which the field is integrated is approximately 1 100 mm^2.

NOTE 4 At 13,56 MHz the approximate inductance is 140 nH and the approximate resistance is 0,3 Ω.

The open circuit calibration factor for the calibration coil 2 is 0,118 V (rms) per A/m (rms) [equivalent to 333 mV (peak-to-peak) per A/m (rms)].

The coil shall be made as a printed coil on printed circuit board (PCB) plated with 35 µm copper. Track width shall be 500 µm with a relative tolerance of ±20 %. The size of the connection pads shall be 1,5 mm × 1,5 mm.

A high impedance oscilloscope probe with an input admittance equivalent to a parallel capacitance C_p < 14 pF and a parallel resistance R_p > 9 kΩ at 13,56 MHz shall be used to measure the (open circuit) voltage induced in the coil.

The high impedance oscilloscope probe ground connection should be as short as possible, less than 20 mm or coaxial connection.

5.4 Test PCD assembly

5.4.1 General

Two Test PCD assemblies are defined:

1) Test PCD assembly 1 for PICCs of "Class 1", "Class 2" and "Class 3" and for PICCs which do not claim compliance with a PICC class;

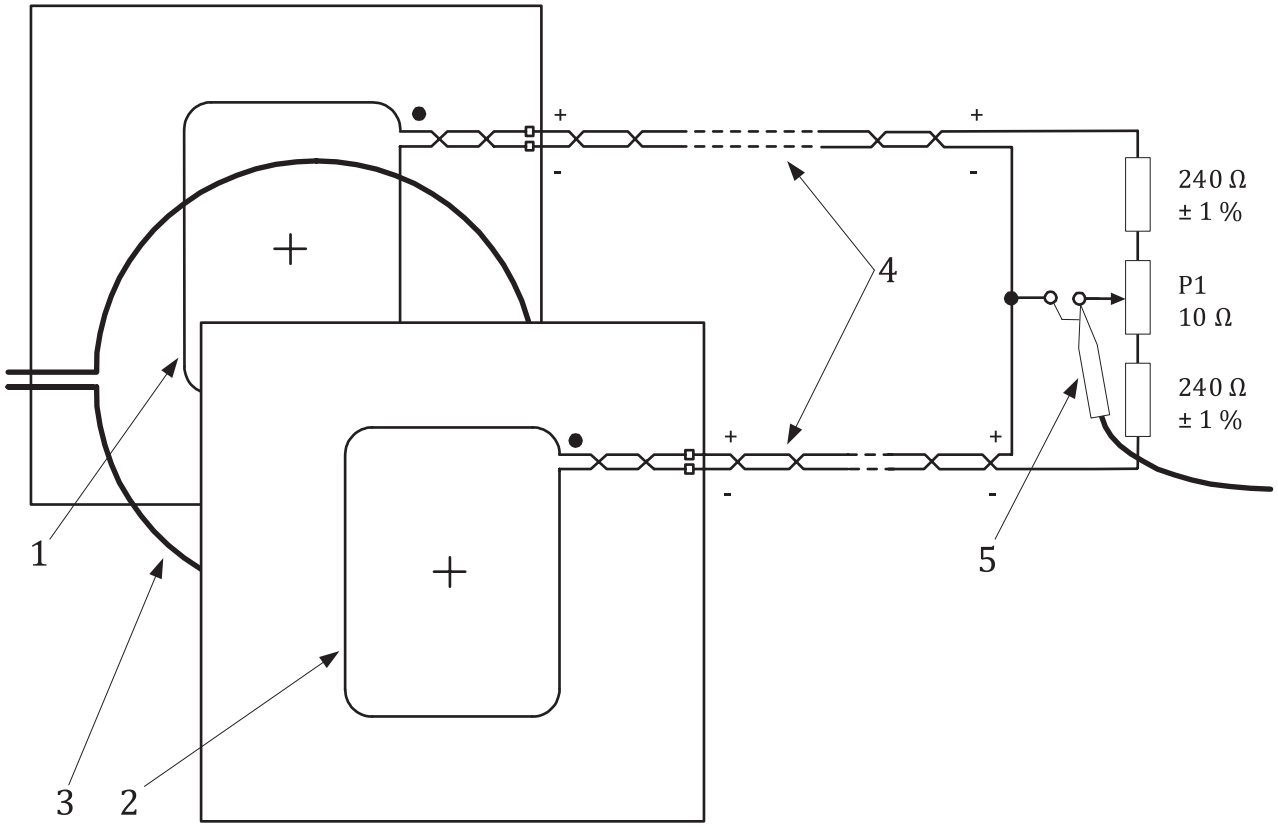
2) Test PCD assembly 2 for PICCs of "Class 4", "Class 5" and "Class 6".

Each Test PCD assembly shall consist of a circular Test PCD antenna and two parallel sense coils, sense coil a and sense coil b, as shown in principle by Figure 2. The sense coils shall be connected such that the signal from one coil is in opposite phase to the other. The 10 Ω potentiometer P1 serves to fine adjust the balance point when the sense coils are not loaded by a PICC or any magnetically coupled circuit. The capacitive load of the probe including its parasitic capacitance shall be less than 14 pF.

The capacitance of the connections and of the oscilloscope probe should be kept to a minimum for reproducibility.

In order to avoid any unintended misalignment in case of an unsymmetrical set-up the tuning range of the potentiometer P1 is only 10 Ω. If the set-up cannot be compensated by the 10 Ω potentiometer P1, the overall symmetry of the set-up should be checked.

The high impedance oscilloscope probe ground connection should be as short as possible, less than 20 mm or coaxial connection.



Key

1 sense coil a
2 sense coil b
3 test PCD antenna
4 identical length twisted pairs or coaxial cables of less than 100 mm
5 probe connected to oscilloscope

Figure 2 — Test set-up (principle)

5.4.2 Test PCD antenna

In Test PCD assembly 1 the Test PCD antenna 1 shall have a diameter of 150 mm.

In Test PCD assembly 2 the Test PCD antenna 2 shall have a diameter of 100 mm.

Each Test PCD antenna construction shall conform to the corresponding drawings in Annex A.

The matching of each Test PCD antenna should be accomplished by using an impedance analyzer or a network analyzer or an LCR meter. If either an impedance analyzer or a network analyzer or an LCR meter is not available, then the matching may be accomplished with the procedure given in Annex B.

5.4.3 Sense coils

In Test PCD assembly 1 the size of the sense coils 1 shall be 100 mm × 70 mm with corner radius 10 mm.

In Test PCD assembly 2 the size of the sense coils 2 shall be 60 mm × 47 mm with corner radius 10 mm.

Each sense coil construction shall conform to the corresponding drawings in Annex C.

5.4.4 Assembly of Test PCD

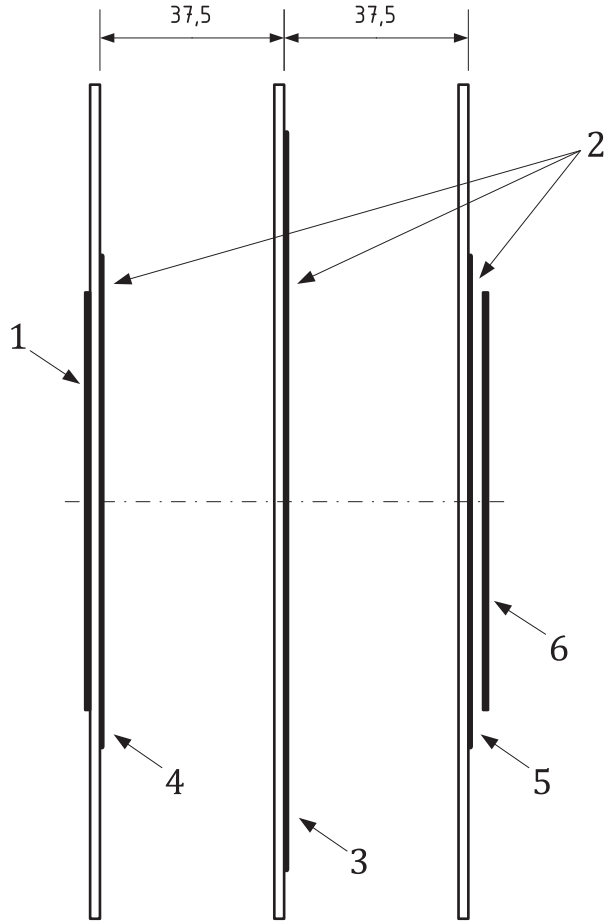
The sense coils 1 and Test PCD antenna 1 shall be assembled parallel and with the sense and antenna coils coaxial and such that the distance between the active conductors is 37,5 mm as shown in Figure 3, Test PCD assembly 1.

The sense coils 2 and Test PCD antenna 2 shall be assembled parallel and with the sense and antenna coils coaxial and such that the distance between the active conductors is 23 mm as shown in Figure 3, Test PCD assembly 2.

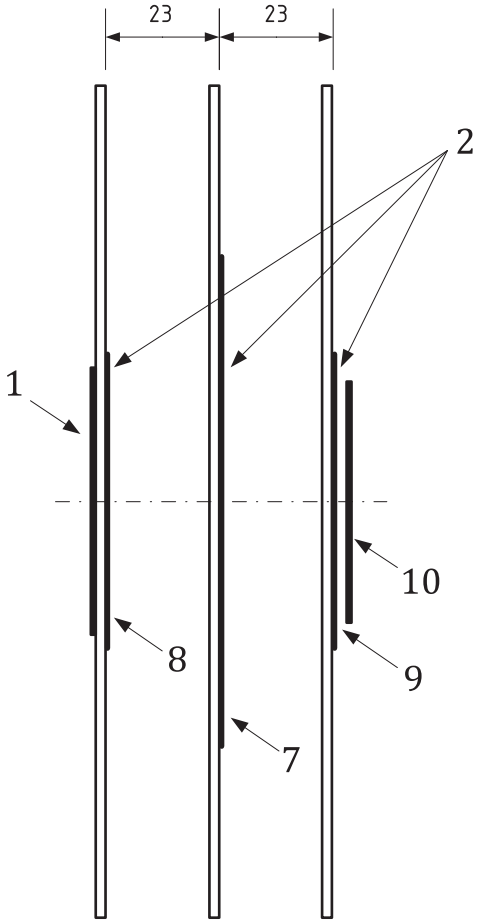
The dimensional tolerance shall be better than ±0,5 mm. The distance between the coil in the DUT and the calibration coil shall be equal with respect to the coil of the Test PCD antenna.

NOTE These distances are chosen to offer a strong and homogenous magnetic field in the DUT position.

Dimensions in millimetres



Test PCD assembly 1



Test PCD assembly 2

Key

1 DUT
2 active conductors
3 test PCD antenna 1
4 sense coil 1a
5 sense coil 1b
6 calibration coil 1
7 test PCD antenna 2
8 sense coil 2a
9 sense coil 2b
10 calibration coil 2

NOTE Drawings are not to scale.

Figure 3 — Test PCD assembly 1 and Test PCD assembly 2

5.5 Reference PICC and Active Reference PICC

5.5.1 General

Two reference equipments are defined to test PCDs:

1) the Reference PICC;

2) the Active Reference PICC.

5.5.2 Reference PICC

5.5.2.1 Introdution

A Reference PICC is defined to test the ability of a PCD to

1) generate a field strength of at least H_{min} and not exceeding H_{max},

2) transmit a modulated signal to a PICC,

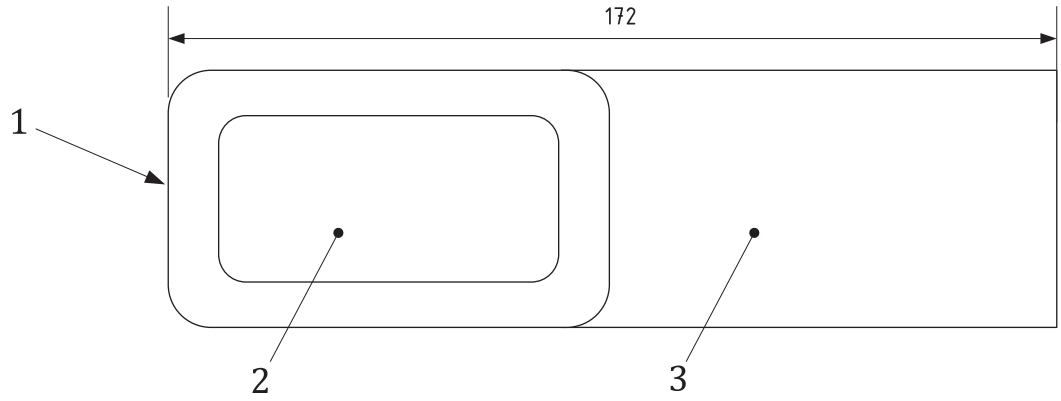
3) be immune against EMD signals from the PICC.

in its operating volume.

5.5.2.2 Dimensions of the Reference PICC

The Reference PICC shall consist of an area containing the coils which has the height and width defined in ISO/IEC 7810 for ID-1 type. An area external to this, containing the circuitry which emulates the required PICC functions, shall be appended in such a way as to allow insertion into the test set-ups and so as to cause no interference to the tests. The dimensions shall be as shown in Figure 4.

Dimensions in millimetres



Key

1 outline ISO/IEC 7810 ID-1

2 location of coils

3 location of circuitry

NOTE Drawing is not to scale.

Figure 4 — Reference PICC dimensions

5.5.2.3 Reference PICC construction

The Reference PICCs coils layouts shall be as defined in D.1. If connectors are used between the coils and the circuitry, those connectors shall have minimal, if any, effect on the RF measurements.

The Reference PICC shall have a circuit diagram as defined in Figure 5 and component values as defined in Table 8.

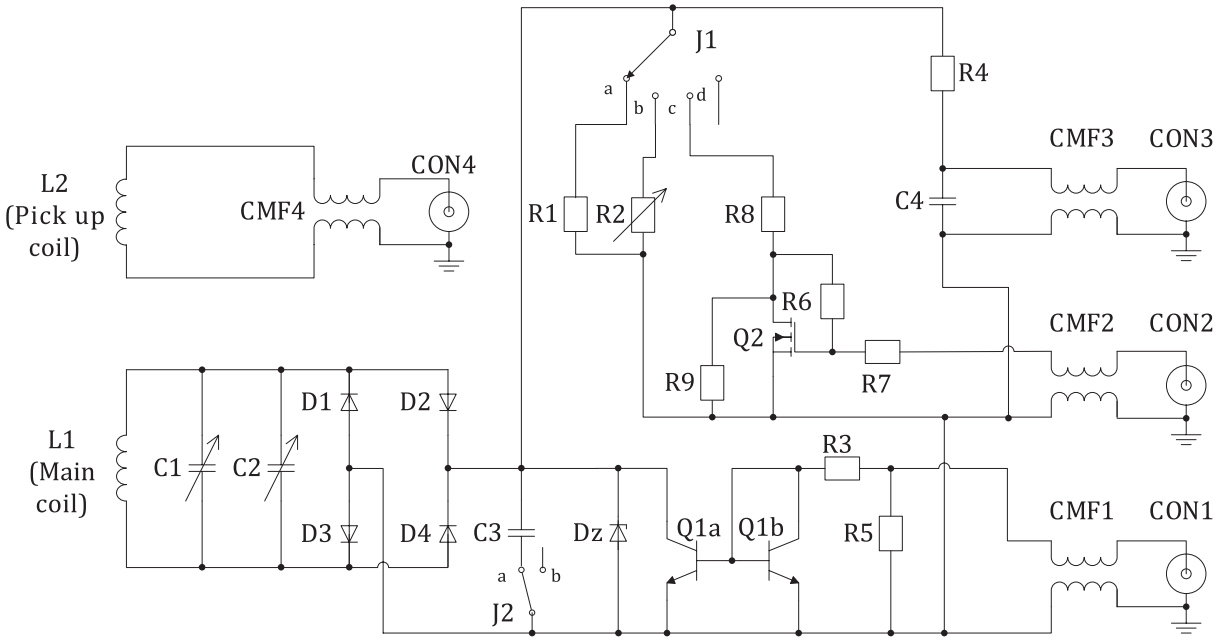


Figure 5 — Reference PICC circuit diagram

NOTE Position 'd' of jumper J1 is RFU.

Table 8 — Reference PICC components list

Component	Value	Component	Value
L1	As defined in D.1	C1	7 pF – 50 pF[b]
L2	As defined in D.1	C2	3 pF – 10 pF[b]
R1	1,8 kΩ	C3	27 pF
R2	0 kΩ – 2 kΩ[a]	C4	1 nF
R3	220 Ω	D1, D2, D3, D4	BAR43S or equivalent[c]
R4	51 kΩ	Dz	BZX84, 15 V or equivalent[c]
R5	51 Ω	Q1a, Q1b	BCV61A or equivalent
R6	500 kΩ	Q2	BSS83 or equivalent
R7	110 kΩ	CMF1, CMF2, CMF3, CMF4	ACM3225-102-2P or equivalent
R8	51 Ω	CON1, CON2, CON3, CON4	RF connector
R9	1,5 kΩ		

[a] A multi-turn potentiometer (turns ≥10) should be used.

[b] Q-factor shall be higher than 100 at 13,56 MHz.

[c] Care should be taken on parameters Cj (Junction capacitance), Cp (Package capacitance), Ls (Series inductance) and Rs (Series resistance) of equivalent diodes. Note that these values may not be available in the datasheet.

At CON1 the load modulation signal shall be applied. The load modulation can be determined in Test PCD assembly. When not used, the load modulation signal generator shall be disconnected or set to 0 V.

With the voltage at CON2 the Reference PICC load can be adjusted until the required DC voltage shows at CON3.

The Reference PICC DC voltage shall be measured at CON3 using a high impedance voltmeter and the connection wires should be twisted or coaxial.

The PCD waveform parameters are picked up at CON4 using a high impedance oscilloscope probe. The high impedance oscilloscope probe ground connection should be as short as possible, less than 20 mm or coaxial connection.

Position 'a' of J2 shall be used for testing bit rates of $f_c/128$, $f_c/64$, $f_c/32$ and $f_c/16$.

Position 'b' of J2 shall be used for testing bit rates of $f_c/8$, $f_c/4$ and $f_c/2$.

5.5.2.4 Reference PICC resonance frequency tuning

The Reference PICC resonance frequency shall be calibrated with the following procedure.

a) Set jumper J1 to position 'a'.

b) Connect the calibration coil directly to a signal generator and the Reference PICC CON3 to a high impedance voltmeter. Connect all the other connectors to the same equipment as used for the tests.

c) Locate the Reference PICC at a distance d = 10 mm above the calibration coil with the axes of the two coils (calibration coil and Reference PICC main coil) being congruent (see Figure 6).

d) Drive the calibration coil with a sine wave set to the desired resonance frequency.

e) Adjust the Reference PICC capacitors C1 and C2 to get maximum DC voltage at CON3.

f) Adjust the signal generator drive level to obtain a DC voltage at CON3 of V_{load} as defined in Table 12.

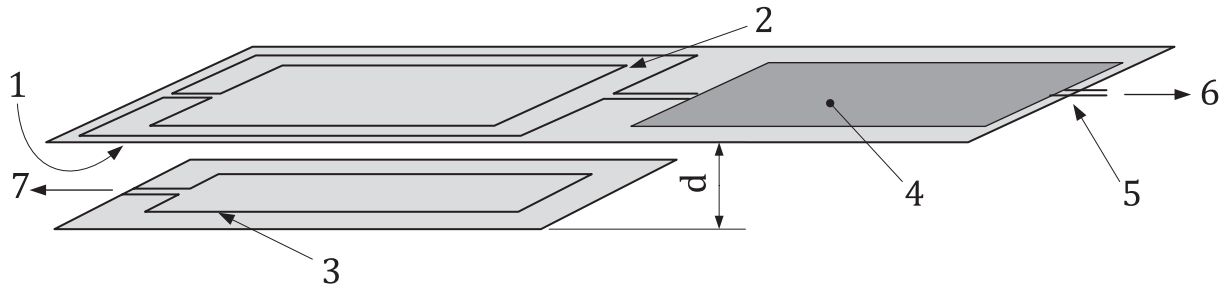
g) Repeat steps e) and f) until the maximum voltage after step e) is V_{load}.

h) Calibrate the Test PCD assembly to produce the H_{min} operating condition on the calibration coil.

i) Place the Reference PICC into the DUT position on the Test PCD assembly. Switch the jumper J1 to position 'b' and adjust R2 to obtain a DC voltage of V_{load} measured at CON3. The operating field condition shall be verified by monitoring the voltage on the calibration coil and adjusted if necessary.

j) Repeat steps b) to g) with the obtained value of R2.

Instead of a signal generator, a vector network analyzer may be used if sufficient power is provided to produce V_{load} at CON3 while reaching the maximum resistive part of the measured complex impedance of the calibration coil.



Key

1 main coil (bottom side)
2 pick up coil (top side)
3 calibration coil (top side)
4 reference PICC circuitry
5 CON3
6 to voltmeter
7 to signal generator
d distance (see procedure above)

NOTE Drawing is not to scale.

Figure 6 — Reference PICC frequency tuning set-up (principle)

5.5.3 Active Reference PICC

5.5.3.1 General

An Active Reference PICC is defined to test the ability of a PCD to receive a load modulation signal from the PICC in its operating volume.

5.5.3.2 Dimensions of the Active Reference PICC

See 5.5.2.2.

5.5.3.3 Active Reference PICC construction

The Active Reference PICCs coils layouts shall be as defined in D.2. If connectors are used between the coils and the circuitry, those connectors shall have minimal, if any, effect on the RF measurements.

The Active Reference PICC shall have a circuit diagram as defined in Figure 7 and component values as defined in Table 9. Input and output connectors are described in Table 10.

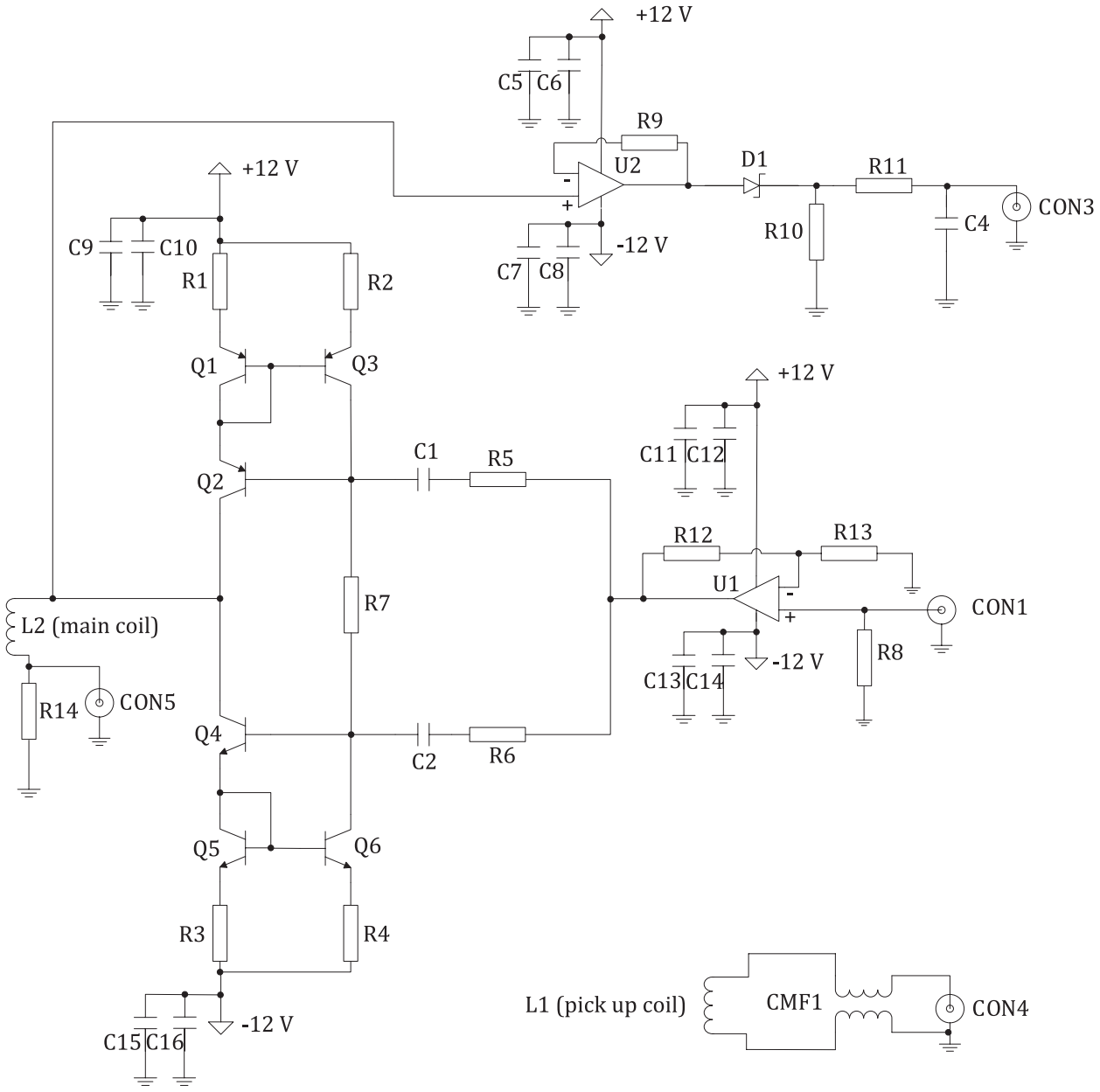


Figure 7 — Active Reference PICC circuit diagram

The resistor R14 should be placed as close as possible to the antenna coil.

Table 9 — Active Reference PICC components list

Component	Value	Power dissipation	Component	Value
R1, R3	10 Ω	235 mW	C1, C2, C4	1 nF
R2, R4, R5, R6	100 Ω	<50 mW	D1	BAT54 or equivalent
R7	1,3 kΩ	250 mW	Q1, Q2, Q3	BD136 or equivalent
R8	50 Ω	250 mW	Q4, Q5, Q6	BD135 or equivalent
R9	1,8 kΩ	<50 mW	U1, U2	THS3091 or equivalent
R10, R11	1,5 kΩ	<50 mW	C5, C7, C9, C11, C13, C15	4,7 µF

Table 9 *(continued)*

Component	Value	Power dissipation	Component	Value
R12, R13	1,2 kΩ	<50 mW	C6, C8, C10, C12, C14, C16	100 nF
R14	0,5 Ω	<50 mW		

Table 10 — Active Reference PICC Input/Output connector description

Connector	Input/Output	Description
CON1	Input	Generation of the Active Reference PICC transmission and unmodulated state US (loading effect).
CON3	Output	Image of the field received when the Active Reference PICC does not transmit (DC voltage).
CON4	Output	PCD carrier extraction.
CON5	Output	Image of the field transmitted by the Active Reference PICC.

The Active Reference PICC received DC voltage shall be measured at CON3 using a high impedance voltmeter and the connection wires should be twisted or coaxial.

High impedance oscilloscope probes shall be used to pick up the signals at CON4 and CON5.

5.6 PICC transmission test setup

5.6.1 General description

The PICC transmission test setup contains:

a) a signal generator with low phase noise, which is used to generate an RF carrier field and to synthesize PCD test commands sent to the PICC under test;

b) the Test PCD assembly;

c) a signal amplitude analyzing device made of a signal acquiring device (e.g. an oscilloscope) and the PCD phase stability analysis tool defined in Annex M.

The signal amplitude analyzing device shall be able to carry out instantaneous phase versus time measurements with fixed frequency, high dynamic range, low measurement uncertainty and high time resolution. See 5.2 for minimum requirements on the amplitude analyzing device.

5.6.2 Phase stability precondition test

5.6.2.1 Purpose

In order to guarantee an RF field with high frequency stability, a phase stability test shall be performed and passed successfully by the PICC transmission test setup. The aim of this precondition test is to verify that the test apparatus used for PICC transmission measurements provides a signal frequency with high stability.

The phase stability test is passed if the maximum phase drift is smaller than ±3° when measured as described in 5.6.2.2.

NOTE The maximum phase drift value of ±3° for the phase stability test covers the measurement uncertainty and noise as well as component fabrication tolerances and temperature dependencies.

5.6.2.2 Test procedure

Perform the following steps to assess the instantaneous phase and compute its maximum phase drift at least at H_{min} and H_{max}.

a) Tune the Reference PICC 1 to 13,56 MHz as described in 5.5.2.4.

b) Adjust the RF power delivered by the signal generator to the Test PCD antenna to the required field strength as measured by the calibration coil.

c) Place the Reference PICC 1 into the DUT position on the Test PCD assembly, set jumper J1 to position 'b' and adjust R2 to obtain a voltage of V_{load} at CON3. Alternatively, jumper J1 may be set to position 'c' and the applied voltage on CON2 is adjusted to obtain a voltage of V_{load} at CON3. In both cases the operating field condition shall be verified by monitoring the voltage in the calibration coil and adjusting if necessary.

d) Record the non-modulated RF carrier signal of the pick up coil for a time period of at least 25 ms.

e) Compute the maximum phase drift from the instantaneous phase using the PCD phase stability analysis tool defined in Annex M. Check if the maximum phase drift of the instantaneous phase is always smaller than ±3° with respect to the initial phase at the very beginning of the time period..

5.6.2.3 Test report

The test report shall contain the maximum phase drift of the PICC transmission test setup and shall state whether the requirements have been fulfilled.

5.7 EMD test setup

5.7.1 General description

The EMD test setup contains the following:

1) a signal generator with low phase noise, which is used to synthesize both an EMD test pattern and PCD test commands sent to the PICC under test;

2) the Test PCD assembly;

3) a signal amplitude analyzing device:

   a) either a signal acquiring device (e.g. oscilloscope) and appropriate computation software;

   b) or a spectrum analyzer (see additional constraints in 5.7.2).

The signal amplitude analyzing device shall be able to carry out power versus time measurements with fixed frequency, fixed bandwidth, high dynamic range, low measurement uncertainty and high time resolution.

The PICC EMD tests may be performed using the RF output signal of a commercial PCD. The PCD EMD test may use a PICC emulator to generate the EMD test pattern.

5.7.2 Computation of power versus time

The beginning of the captured signal shall be windowed by a Bartlett window of exactly two subcarrier cycles. Fourier transformation of these windowed samples produces one power value. By shifting the Bartlett window by steps of $1/f_c$ from the beginning to the end of the captured signal, the desired power versus time result is finally computed.

NOTE The resulting 3 dB bandwidth of the above described window is 531 kHz and its noise equivalent bandwidth amounts to 843 kHz.

The computation of the power versus time shall be performed at $f_c + f_s$ and $f_c - f_s$, using a scaling such that a pure sinusoidal signal results in its peak magnitude. An example of computation is provided in Annex J.

In case of using a spectrum analyzer, the analyzer shall have at least an equivalent analysis bandwidth. It shall pass the noise floor precondition test, as defined in 5.7.3, and there shall be some additional margin of $10/f_c$ on $t_{E,\,PICC}$ requirement and no spikes above the EMD limit.

5.7.3 Noise floor precondition test

5.7.3.1 Purpose

In order to ensure a high dynamic range and sufficient sensitivity, a noise floor measurement shall be performed and passed successfully by the EMD test setup. The aim of this precondition test is to verify that the test apparatus used for EMD level measurement satisfies a minimum noise requirement.

The noise floor test is passed if the noise standard deviation is at least three times smaller than the EMD limit $V_{E,\,PICC}$, when measured as described in 5.7.3.2.

The noise standard deviation is determined by calculating the root-mean-square value of the results of the Fourier transformation, as described in 5.7.2.

NOTE This noise floor can be obtained either with a 14-bit digitizer at a sampling rate of 100 million samples per second or with an 8-bit digital oscilloscope at sampling rate of 1 000 million samples per second.

5.7.3.2 Procedure

Perform the following steps to assess the noise floor at least at H_{min} and H_{max}.

a) Tune the Reference PICC 1 to 13,56 MHz.

b) Adjust the RF power delivered by the signal generator to the Test PCD antenna to the required field strength as measured by the calibration coil.

c) Place the Reference PICC 1 into the DUT position on the Test PCD assembly, set jumper J1 to position 'b' and adjust R2 to obtain a voltage of V_{load} at CON3. Alternatively, jumper J1 may be set to position 'c' and the applied voltage on CON2 is adjusted to obtain a voltage of V_{load} at CON3. In both cases the operating field condition shall be verified by monitoring the voltage in the calibration coil and adjusted if necessary.

d) Record the signal of the sense coils for a time period of at least 250 µs.

e) Compute the noise standard deviations at $f_c + f_s$ and $f_c - f_s$ using suitable computer software, as e.g. the one given in Annex J. Check if these noise standard deviations are three times smaller than $V_{E,\,PICC}$.

5.7.3.3 Test report

The test report shall state the noise standard deviations at $f_c + f_s$ and $f_c - f_s$ and shall state whether the requirements have been fulfilled.

6 Test of ISO/IEC 14443-1 parameters

6.1 PCD tests

6.1.1 Alternating magnetic field

6.1.1.1 Purpose

This test determines that the PCD generates a field not higher than the average value specified in ISO/IEC 14443-1 for each PICC class, in any possible PICC position.

6.1.1.2 Procedure

Apply the following procedure with each Reference PICC:

a) Tune the Reference PICC to 19 MHz as described in 5.5.2.4 steps a) to g).

b) Calibrate the Test PCD assembly (see Table 11) to produce the average field value specified in ISO/IEC 14443-1:2018, 4.4 on the calibration coil.

c) Place the Reference PICC into the DUT position on the Test PCD assembly (see Table 11). Switch the jumper J1 to position 'b' and adjust R2 to obtain a DC voltage of 3 V measured at CON3. The operating field condition shall be verified by monitoring the voltage on the calibration coil and adjusted if necessary.

   WARNING — R2 value should be between $R2_{min}$ and $R2_{max}$ as defined in Table 11.

d) Position the Reference PICC in any possible PICC position. The DC voltage at CON3 shall not exceed 3 V.

   WARNING — If the PCD field is not continuously emitted, the DC voltage measurement shall be done when the PCD field is present.

e) If exceeded, use the same conversion factor to measure the maximum and average DC voltage and convert in field strength to check the maximum and the average field values specified in ISO/IEC 14443-1:2018, 4.4 in a 30 s period.

Table 11 — R2 value for alternating magnetic field test

Reference PICC	$R2_{min}$	$R2_{max}$	Test PCD assembly
1	56 Ω	68 Ω	Test PCD assembly 1
2	97 Ω	119 Ω	Test PCD assembly 1
3	100 Ω	122 Ω	Test PCD assembly 1
4	84 Ω	102 Ω	Test PCD assembly 2
5	76 Ω	92 Ω	Test PCD assembly 2
6	88 Ω	108 Ω	Test PCD assembly 2

6.1.1.3 Test report

The test report shall give the DC voltage measured at CON3 for each Reference PICC. The test result is PASS only if the PCD meets the requirements given by the last two steps of the procedure in 6.1.1.2 for each Reference PICC, otherwise the test result is FAIL.

6.2 PICC tests

6.2.1 Alternating magnetic field

6.2.1.1 Purpose

The purpose of this test is to check the behavior of the PICC in relation to alternating magnetic field exposure at 13,56 MHz.

6.2.1.2 Apparatus

The Test PCD assembly shall be used to produce the alternating magnetic field.

6.2.1.3 Procedure

The procedure is as follows.

a) In accordance with the PICC class, adjust the RF power delivered by the signal generator to the Test PCD antenna to a field strength of the average level specified in ISO/IEC 14443-1:2018, 4.4 as measured by the calibration coil.

b) Place the PICC under test in the DUT position and readjust immediately the RF drive into the Test PCD antenna to the required field strength if necessary.

c) After 5 min, remove the PICC from the DUT position for at least 5 s.

d) In accordance with the PICC class, adjust the RF power delivered by the signal generator to the Test PCD antenna to a field strength of the maximum level specified in ISO/IEC 14443-1:2018, 4.4 as measured by the calibration coil.

e) Place the PICC under test in the DUT position and readjust immediately the RF drive into the Test PCD antenna to the required field strength if necessary.

f) Apply for 5 min an ASK 100 % modulation to this field with the following duty cycle:

   1) 5 s at 0 A/m (rms);

   2) 25 s at the maximum level specified in ISO/IEC 14443-1:2018, 4.4, in accordance with the PICC class.

g) Wait for 30 s at 0 A/m (rms).

h) Check that the PICC operates as intended. This check should be done at least by successfully executing PICC transmission test defined in 7.2.1.

6.2.1.4 Test report

The test report shall state whether or not the PICC operates as intended.

6.2.2 Void

The static electricity test, which was defined in former editions of this document, is no longer present.

6.3 PXD tests

PCD and PICC tests shall be applied as follows:

1) when the PXD is in PCD Mode, tests defined in 6.1 shall be applied;

2) when the PXD is in PICC Mode, tests defined in 6.2 shall be applied.

In automatic mode alternation, the PXD may be forced into the required mode.

7 Test of ISO/IEC 14443-2 parameters

7.1 PCD tests

All the PCD tests described below will be done in the operating volumes as defined by the PCD manufacturer for each supported PICC class.

All PCD tests of ISO/IEC 14443-2 parameters shall be performed using Reference PICCs 1, 2 and 3 and optionally other Reference PICCs corresponding to the optional PICC classes supported by the PCD, with the relevant parameters and Test PCD assembly as defined in Table 12.

Table 12 — PICC classes parameters

PICC class	Reference PICC	H_{min} test			H_{max} test		Test PCD assembly
		V_{load}	$R2_{min}$	$R2_{max}$	$R2_{min}$	$R2_{max}$	
1	1	4,5 V	647 Ω	791 Ω	76 Ω	94 Ω	Test PCD assembly 1
2	2	4,5 V	1 030 Ω	1 260 Ω	112 Ω	138 Ω	Test PCD assembly 1
3	3	4,5 V	1 080 Ω	1 320 Ω	117 Ω	143 Ω	Test PCD assembly 1
4	4	4,5 V	990 Ω	1 210 Ω	99 Ω	121 Ω	Test PCD assembly 2
5	5	4,5 V	960 Ω	1 170 Ω	103 Ω	127 Ω	Test PCD assembly 2
6	6	4,5 V	700 Ω	900 Ω	117 Ω	143 Ω	Test PCD assembly 2

The PCD should be tested with a V_{load} of 6 V for Reference PICC 1 to ensure interoperability with PICCs compliant with the V_{load} requirement of ISO/IEC 10373-6:2016.

7.1.1 PCD field strength

7.1.1.1 Purpose

This test measures the field strength produced by a PCD in its operating volume. The maximum and minimum field strength values to be used with each Reference PICC are given in ISO/IEC 14443-2:2020, Table 1.

NOTE The test takes account of PICC loading of the PCD.

7.1.1.2 Procedure

7.1.1.2.1 Procedure for H_{max} test

a) Tune the Reference PICC to 19 MHz as described in 5.5.2.4 steps a) to g).

b) Calibrate the Test PCD assembly to produce the H_{max} operating condition on the calibration coil.

c) Place the Reference PICC into the DUT position on the Test PCD assembly. Switch the jumper J1 to position 'b' and adjust R2 to obtain a DC voltage of 3 V measured at CON3. Alternatively, jumper J1 may be set to position 'c' and the applied voltage on CON2 is adjusted to obtain a DC voltage of 3 V at CON3. In both cases, the operating field condition shall be verified by monitoring the voltage on the calibration coil and adjusted if necessary.

WARNING — R2 value should be between $R2_{min}$ and $R2_{max}$ as defined in H_{max} test columns in Table 12. Check this range at least once before using the alternative method.

d) Position the Reference PICC within the defined operating volume of the PCD under test. The DC voltage at CON3 shall not exceed 3 V.

WARNING — If the PCD field is not continuously emitted, the DC voltage measurement shall be done when the PCD field is present.

7.1.1.2.2 Procedure for H_{min} test

a) Tune the Reference PICC to 13,56 MHz as described in 5.5.2.4.

b) Place the Reference PICC into the DUT position on the Test PCD assembly producing the H_{min} operating condition on the calibration coil. Check that the jumper J1 is set to position 'b' and that a DC voltage of V_{load} as defined in Table 12 is measured at CON3. Alternatively, the jumper J1 may be set to position 'c' and the voltage on CON2 is adjusted to obtain a DC voltage of V_{load} as defined in Table 12 at CON3. In both cases, the operating field condition shall be verified by monitoring the voltage on the calibration coil and adjusted if necessary.

WARNING — R2 value should be between $R2_{min}$ and $R2_{max}$ as defined in H_{min} test columns in Table 12. Check this range at least once before using the alternative method.

c) Position the Reference PICC within the defined operating volume of the PCD under test. The DC voltage at CON3 shall exceed V_{load} as defined in Table 12.

WARNING — If the PCD field is not continuously emitted, the DC voltage measurement shall be done when the PCD field is present.

d) For Reference PICC 1 repeat:

1) step b) with a DC voltage V_{load} of 6 V measured at CON3 instead of V_{load} as defined in Table 12, and

2) step c), where the DC voltage at CON3 shall exceed 5,3 V [corresponding to H_{min} – 0,2 A/m (rms)], instead of V_{load} as defined in Table 12.

7.1.1.3 Test report

The test report shall confirm the operating volume in which the DC voltage measured at CON3 for R2 or variable load resistor adjusted to H_{min} and H_{max} field strength fulfills the requirements defined in step d) of 7.1.1.2.1 and in step c) and step d) of 7.1.1.2.2.

7.1.2 Void

PCD field strength supporting operation with PICC Class 1, which was defined in former editions of this document, is no longer present.

7.1.3 Void

Power transfer PCD to PICC, which was defined in former editions of this document, is no longer present.

7.1.4 Modulation index *m* and waveform

7.1.4.1 Purpose

This test is used to determine the index of modulation of the PCD field as well as the rise and fall times and the overshoot values as defined in ISO/IEC 14443-2 for all supported PCD to PICC bit rates.

7.1.4.2 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. Determine the modulation index *m* and waveform characteristics using the analysis tool defined in Annex E.

b) Tune the Reference PICC to 16,5 MHz as described in 5.5.2.4 steps a) to g) and switch the jumper J1 to position 'c'.

c) Switch the jumper J2 to position 'a' or to position 'b' according to the tested PCD to PICC bit rate and 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 CON3 of V_{load} as defined in Table 12. 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 (requirements see 5.2) is below 1 V (peak-to-peak), use an alternative pick up coil to determine the waveform characteristic. 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 modulation index *m* and waveform characteristic from the voltage at CON4 or at the alternative pick up coil using the analysis tool defined in Annex E.

g) Repeat steps c) to f) for various positions within the operating volume and all supported PCD to PICC bit rates.

NOTE 1 The selected position of the calibration coil within the operating volume is not expected to affect the results.

NOTE 2 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).

7.1.4.3 Test report

The test report shall give the measured modulation index *m* of the PCD field, the rise and fall times and overshoot values, within the defined operating volume in unloaded and loaded conditions.

7.1.5 Phase stability test

7.1.5.1 Purpose

The purpose of this test is to determine whether the PCD provides f_c with a phase drift within the limit defined in ISO/IEC 14443-2.

7.1.5.2 Test procedure

a) Tune the Reference PICC 1 to 13,56 MHz as described in 5.5.2.4.

b) Place the Reference PICC 1 at an arbitrary position in the PCD operating volume, set jumper J1 to position 'b' and adjust R2 to obtain a voltage of V_{load} (see Table 12) at CON3. Alternatively, jumper J1 may be set to position 'c' and the applied voltage on CON2 is adjusted to obtain a voltage of V_{load} at CON3.

c) Record the signal of the pick up coil for a time period of at least 25 ms.

d) Compute the maximum phase drift from the instantaneous phase using the PCD phase stability analysis tool defined in Annex M. Check if the maximum phase drift of the instantaneous phase is smaller than the limit specified in ISO/IEC 14443-2:2020, 8.2.5.2.

7.1.5.3 Test report

The test report shall contain the maximum phase drift of the PCD.

The used test conditions shall be mentioned in the test report.

7.1.6 Load modulation reception for PICC to PCD bit rates of $f_c/128$, $f_c/64$, $f_c/32$ and $f_c/16$

7.1.6.1 Purpose

This test is used to verify that a PCD correctly detects the load modulation of a PICC which conforms to ISO/IEC 14443-2 for PICC to PCD bit rates of $f_c/128$, $f_c/64$, $f_c/32$ and $f_c/16$, if supported.

In order to limit the testing time, only the highest PICC to PCD bit rate corresponding to every supported subcarrier f_s shall be tested.

NOTE In future revisions of ISO/IEC 10373-6, it is intended to extend this test to cover also PICC to PCD bit rates of $f_c/8$, $f_c/4$ and $f_c/2$.

7.1.6.2 General description

For the load modulation reception test the Active Reference PICC uses the principle of active load modulation to modify the field generated by the PCD.

The active load modulation signal of the Active Reference PICC shall be strictly synchronous to the PCD generated field. The pick up coil voltage on CON4, an alternative pick up coil (see 7.1.4.2, step e)), the main coil current on CON5 or any other mean may be used to synchronize the Active Reference PICC active load modulation signal generating device to the PCD RF field. It is recommended to perform synchronization during time periods when neither the PCD nor the Active Reference PICC modulates the RF field. During the Active Reference PICC response, the Active Reference PICC phase reference shall not vary more than 3° from the $\emptyset_{LM,\ INIT}$ value when it was synchronized; if synchronization is done much before the Active Reference PICC response, care shall be taken to limit the possible phase drift until the start of the Active Reference PICC response.

The active load modulation signal shall be generated such that the load modulation signal envelope observed at the load modulation test circuit of Figure 2 has a sinusoidal shape without phase jumps.

7.1.6.3 Precondition test

Perform the following steps to verify the synchronous functionality of the Active Reference PICC:

a) Place the Active Reference PICC in the DUT position of the Test PCD assembly.

b) Calibrate the Test PCD assembly to produce any field strength operating condition in the range of $[H_{min}, H_{max}]$ on the calibration coil.

c) Adjust the load of the Active Reference PICC to the equivalent load defined for the Reference PICC (with V_{load} DC voltage at Reference PICC CON3 as defined in Table 12 as setup at H_{min}) having the same antenna class by applying a constant carrier signal to connector CON1, adjusting first its phase in order to achieve destructive interference and then its amplitude. The operating field condition shall be verified by monitoring the voltage on the calibration coil and adjusted if necessary. Note the Active Reference PICC CON3 DC voltage corresponding to the field condition.

d) Send a command with the Test PCD assembly.

e) The Active Reference PICC shall respond strictly synchronously i.e., with a theoretical $\emptyset_{LM,\ INTER}$ of 0° and a constant $\emptyset_{LM,\ INIT}$ between multiple responses. The Active Reference PICC shall respond with a 256 byte frame at a PICC to PCD bit rate of $f_c/128$.

f) Display the whole Active Reference PICC response of both, calibration coil and load modulation test circuit, including at least 200 carrier periods before the first and 20 carrier periods after the last modulation of the Active Reference PICC on the signal acquiring device and store the sampled data of calibration coil and load modulation test circuit in separate files for analysis by the PICC amplitude and phase drift analysis tool defined in Annex N.

g) Repeat steps d) to f) 24 times.

The precondition test is PASS when the Active Reference PICC response $\emptyset_{LM,\ INTER}$ is less than ±3° and its $\emptyset_{LM,\ INIT}$ maximum variation is less than ±3° over 24 responses.

7.1.6.4 Test conditions

7.1.6.4.1 General

This test shall be done at ambient, minimum and maximum temperatures. If minimum and maximum temperatures are not provided in PCD test conditions, load modulation reception test shall be performed only at ambient temperature.

Test conditions are communication signal interface and PICC to PCD bit rate dependent defined and are described in the following two subclauses. Figure 8 to Figure 17 illustrate the test conditions.

1) The top plot shows the Active Reference PICC carrier signal:

   a) Active Reference PICC carrier transmission, equivalent to state MS1, is shown as a black colored bar,

   b) inverse Active Reference PICC carrier transmission, equivalent to state MS2, is shown as grey colored bar, and

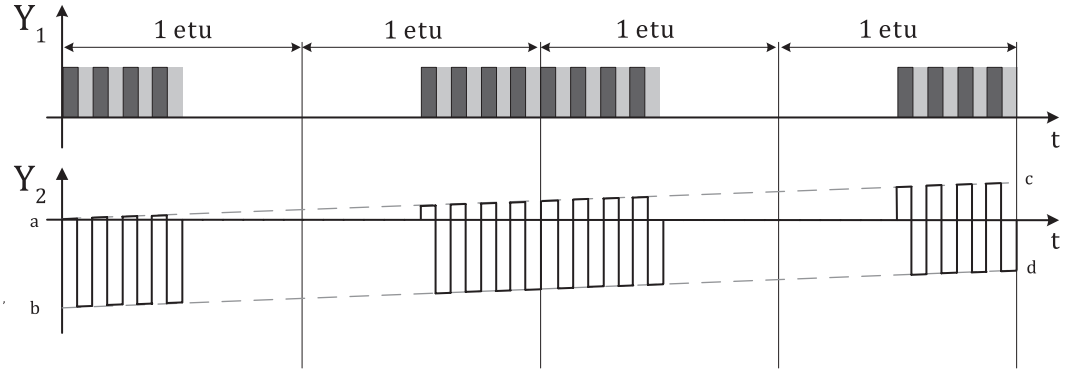
   c) no Active Reference PICC carrier transmission, equivalent to state US, is shown in white color.

2) The bottom plot shows the phase signal also including the interstate phase drift (grey dashed line) over time.

7.1.6.4.2 Test conditions for Type A for a PICC to PCD bit rate of $f_c/128$

Four phase drift conditions are defined within limits defined in ISO/IEC 14443-2.

1) Phase drift condition A1: Constant interstate phase drift of $\emptyset_{LM,\ INTER,\ PCD}$ over the complete Active Reference PICC response, active load modulation transmission during state MS1 and active load modulation transmission with the inverse phase of MS1 during state MS2. For an example see Figure 8.

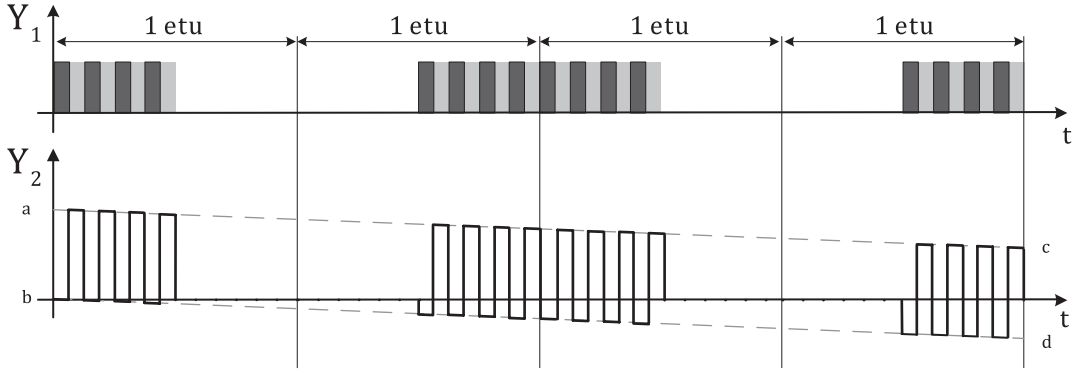


Key

Y_1	amplitude	a	$Ø_{LM, INIT}$
Y_2	phase (degree)	b	$Ø_{LM, INIT} - 180°$
		c	$Ø_{LM, INIT} + Ø_{LM, INTER, PCD}$
		d	$Ø_{LM, INIT} - 180° + Ø_{LM, INTER, PCD}$

Figure 8 — Example of phase drift condition A1 for Type A at a PICC to PCD bit rate of $f_c/128$

2) Phase drift condition A2: Constant interstate phase drift of $-Ø_{LM, INTER, PCD}$ over the complete Active Reference PICC response, active load modulation transmission during state MS1 and active load modulation transmission with the inverse phase of MS1 during state MS2. For an example see Figure 9.



Key

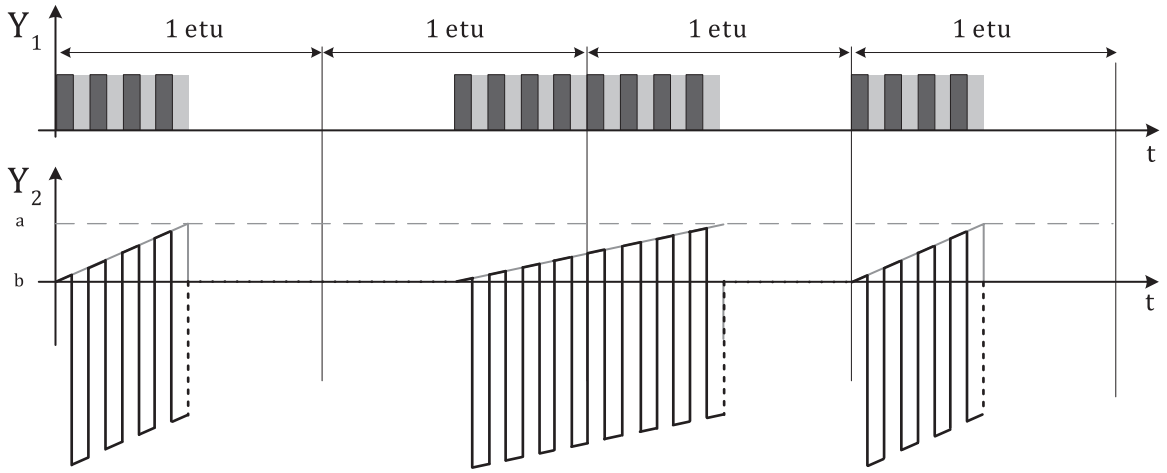
Y_1	amplitude	a	$Ø_{LM, INIT} + 180°$
Y_2	phase (degree)	b	$Ø_{LM, INIT}$
		c	$Ø_{LM, INIT} + 180° - Ø_{LM, INTER, PCD}$
		d	$Ø_{LM, INIT} - Ø_{LM, INTER, PCD}$

Figure 9 — Example of phase drift condition A2 for Type A at PICC to PCD bit rate of $f_c/128$

3) Phase drift condition A3: Constant interstate phase drift of $Ø_{LM, INTER, PCD}$ during sequence D and E defined in ISO/IEC 14443-2:2020, 8.2.6.1. For an example see Figure 10.

   a) Active load modulation transmission during MS1 and active load modulation transmission with the inverse phase of MS1 during state MS2.

   b) If sequence D directly follows sequence E, the constant interstate phase drift shall be $Ø_{LM, INTER, PCD}$ over both sequences.



Key

Y_1 amplitude

Y_2 phase (degree)

a $Ø_{LM, INIT} + Ø_{LM, INTER, PCD}$

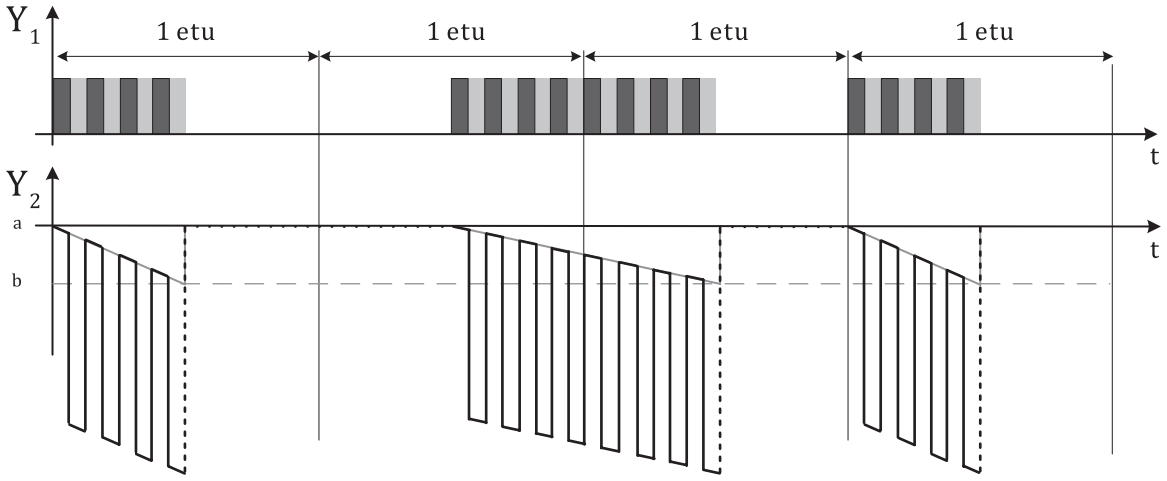
b $Ø_{LM, INIT}$

Figure 10 — Example of phase drift condition A3 for Type A at PICC to PCD bit rate of $f_c/128$

4) Phase drift condition A4: Constant interstate phase drift of $-Ø_{LM, INTER, PCD}$ during sequence D and E defined in ISO/IEC 14443-2:2020, 8.2.6.1. For an example see Figure 11.

   a) Active load modulation transmission during MS1 and active load modulation transmission with the inverse phase of MS1 during state MS2.

   b) If sequence D directly follows sequence E, the constant interstate phase drift shall be $-Ø_{LM, INTER, PCD}$ over both sequences.



Key

Y_1 amplitude

Y_2 phase (degree)

a $Ø_{LM, INIT}$

b $Ø_{LM, INIT} - Ø_{LM, INTER, PCD}$

Figure 11 — Example of phase drift condition A4 for Type A at PICC to PCD bit rate of $f_c/128$

All of these phase drift conditions shall be tested at:

— $V_{LMA, min, PCD}$ and $V_{LMA, max, PCD}$

— 24 different $Ø_{LM, INIT}$ in steps of 15°.

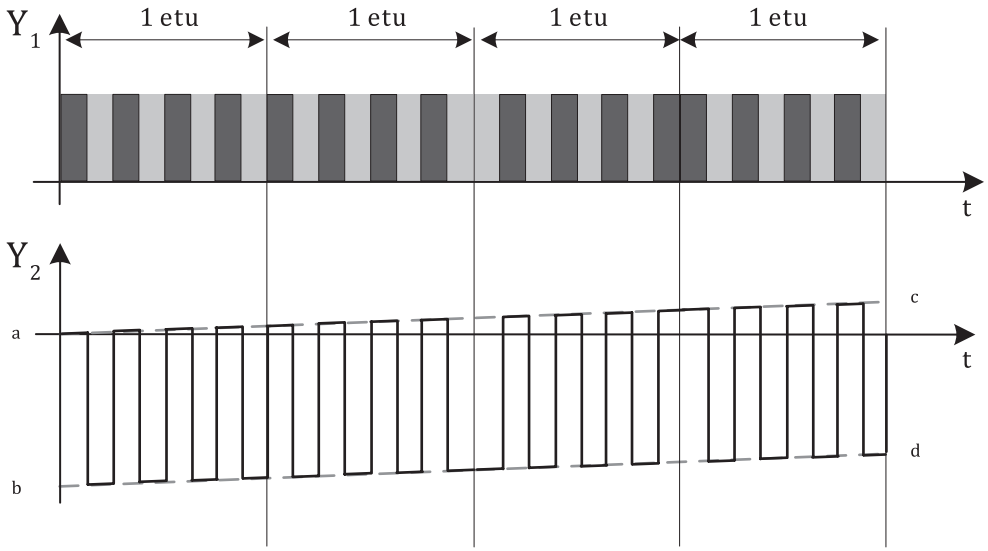
This results in 192 tests per Active Reference PICC.

NOTE Further $\emptyset_{LM,\ INIT}$ phases may be tested.

7.1.6.4.3 Test conditions for Type B for a PICC to PCD bit rate of $f_c/128$ and for Type A and Type B for optional PICC to PCD bit rates of $f_c/64$, $f_c/32$ and $f_c/16$

Two phase drift conditions are defined within limits defined in ISO/IEC 14443-2.

1) Phase drift condition B1: Constant interstate phase drift of $\emptyset_{LM,\ INTER,\ PCD}$ over the complete Active Reference PICC response, active load modulation transmission during state MS1 and active load modulation transmission with the inverse phase of MS1 during state MS2. For an example see Figure 12.



Key

Y_1 amplitude

Y_2 phase (degree)

a $\emptyset_{LM,\ INIT}$

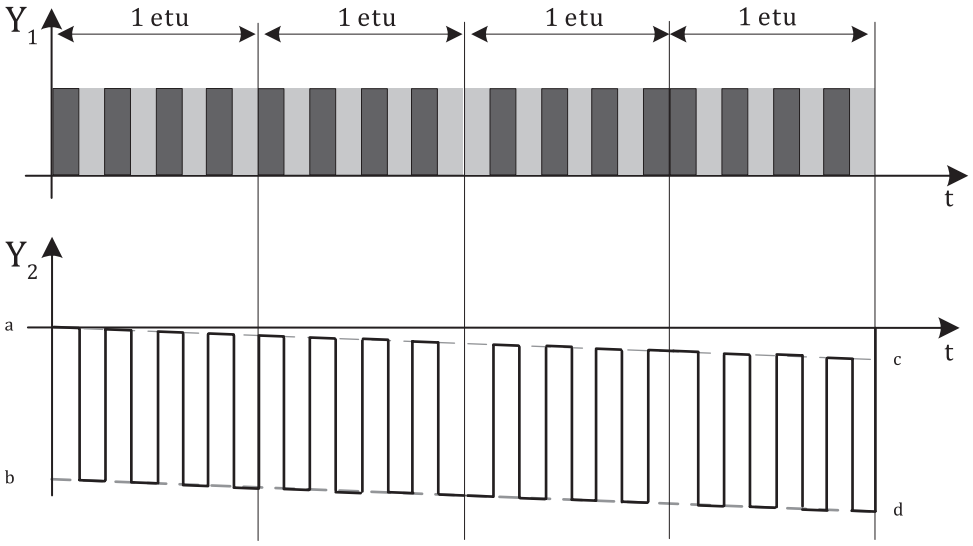
b $\emptyset_{LM,\ INIT} - 180°$

c $\emptyset_{LM,\ INIT} + \emptyset_{LM,\ INTER,\ PCD}$

d $\emptyset_{LM,\ INIT} - 180° + \emptyset_{LM,\ INTER,\ PCD}$

Figure 12 — Example of phase drift condition B1 at PICC to PCD a bit rate of $f_c/64$

2) Phase drift condition B2: Constant interstate phase drift of $-\emptyset_{LM,\ INTER,\ PCD}$ over the complete Active Reference PICC response, active load modulation transmission during state MS1 and active load modulation transmission with the inverse phase of MS1 during state MS2. For an example see Figure 13.



Key

Y_1	amplitude	a	$Ø_{LM, INIT}$
Y_2	phase (degree)	b	$Ø_{LM, INIT} - 180°$
		c	$Ø_{LM, INIT} - Ø_{LM, INTER, PCD}$
		d	$Ø_{LM, INIT} - 180° - Ø_{LM, INTER, PCD}$

Figure 13 — Example of phase drift condition B2 at PICC to PCD a bit rate of $f_c/64$

All of these phase drift conditions shall be tested at:

— $V_{LMA, min, PCD}$ and $V_{LMA, max, PCD}$

— 24 different $Ø_{LM, INIT}$ in steps of 15°.

This results in 96 tests per each combination of PICC to PCD bit rate, communication signal interface and Active Reference PICC.

NOTE Further $Ø_{LM, INIT}$ phases may be tested.

7.1.6.5 Test procedure 1

The PCD shall operate under the conditions defined in 7.1.6.4 after the selection of that PICC to PCD bit rate. The PCD shall correctly react to a received PICC response at the selected PICC to PCD bit rate.

a) Place the Active Reference PICC in the DUT position of the Test PCD assembly.

b) Calibrate the Test PCD assembly to produce H_{min} in accordance with the Active Reference PICC class.

c) Adjust the load of the Active Reference PICC to the equivalent load defined for the Reference PICC (with V_{load} DC voltage at Reference PICC CON3 as defined in Table 12 as setup at H_{min}) having the same antenna class by applying a constant carrier signal to connector CON1, adjusting first its phase in order to achieve destructive interference and then its amplitude. The operating field condition shall be verified by monitoring the voltage on the calibration coil and adjusted if necessary. Note the Active Reference PICC CON3 DC voltage corresponding to the field condition.

d) Adjust the Active Reference PICC input signal at CON1 to get $V_{LMA, min, PCD}$ associated with the noted field strength and PICC class as specified in ISO/IEC 14443-2.

NOTE The V_{LMA} is measured as described in 7.2.1.

e) Select an applicable phase drift condition defined in 7.1.6.4.

f) Produce a response at a PICC to PCD bit rate of $f_c/128$ and record the initial phase $\emptyset_{LM,\ INIT}$ using the PICC amplitude and phase drift analysis tool of Annex N.

g) Place the Active Reference PICC in a position in the PCD operating volume with the same field strength condition as setup in step c) by measuring CON3 DC voltage. If this field strength condition cannot be achieved, measure the maximum field strength which can be achieved with a Reference PICC (with V_{load} DC voltage at Reference PICC CON3 as defined in Table 11) having the same antenna class then redo the procedure steps a), b) and d) using this maximum achievable field strength in steps b) and d).

h) The PCD shall correctly detect the Active Reference PICC response.

i) Repeat step h) at least for all mandatory $\emptyset_{LM,\ INIT}$ values defined in 7.1.6.4.

j) Repeat steps h) and i) for all applicable phase drift conditions defined in 7.1.6.4.

k) Repeat steps a) and d) to j) for the highest PICC to PCD bit rate corresponding to every supported subcarrier f_s.

l) Repeat steps a) and d) to k) at least for all mandatory PCD V_{LMA} values defined in 7.1.6.4.

m) Repeat steps a), b) and d) to l) at least for all field strength conditions defined in Table 5.

Any tested position where the load modulation reception test is FAIL, shall be considered out of the operating volume.

7.1.6.6 Test procedure 2

a) Place the Active Reference PICC in the DUT position of the Test PCD assembly.

b) Calibrate the Test PCD assembly to produce H_{min} in accordance with the Active Reference PICC class.

c) Minimize the load of the Active Reference PICC by applying no constant carrier signal to connector CON1. The operating field condition shall be verified by monitoring the voltage on the calibration coil and adjusted if necessary. Note the Active Reference PICC CON3 DC voltage corresponding to the field condition.

d) Adjust the Active Reference PICC input signal at CON1 to produce $f_c - f_s$ and $f_c + f_s$ sidebands amplitudes of $V_{LMA,\ min,\ PCD}$ associated with H_{min} and PICC class as specified in ISO/IEC 14443-2.

e) Select phase drift condition A1 or B1 defined in 7.1.6.4, depending on the PICC communication signal interface and the PICC to PCD bit rate.

f) Produce a response and record the initial phase $\emptyset_{LM,\ INIT}$ using the PICC amplitude and phase drift analysis tool of Annex N.

g) Place the Active Reference PICC in a position in the PCD operating volume with the same field strength condition as setup in step c) by measuring CON3 DC voltage.

h) The PCD shall correctly detect the Active Reference PICC response.

i) Repeat step h) at least for all mandatory $\emptyset_{LM,\ INIT}$ values defined in 7.1.6.4.

j) With $f_c + f_s$ sideband amplitude of PCD V_{LMA} as adjusted in step d), repeat steps h) and i) for $f_c - f_s$ sideband amplitude increased by 2,0 dB, 4,0 dB, 6,0 dB, 8,0 dB and 10,0 dB.

k) With $f_c - f_s$ sideband amplitude of PCD V_{LMA} as adjusted in step d), repeat steps h) and i) for $f_c + f_s$ sideband amplitude increased by 2,0 dB, 4,0 dB, 6,0 dB, 8,0 dB and 10,0 dB.

l) Repeat steps h) to k) for the highest supported PICC to PCD bit rate corresponding to every subcarrier f_s.

Any tested position where the load modulation reception test is FAIL, shall be considered out of the operating volume.

7.1.6.7 Test report

The test report shall confirm the intended operation at the mandatory PICC to PCD bit rate of $f_c/128$ for Test procedure 1 and Test procedure 2. For PCDs supporting one or more of the optional PICC to PCD bit rates the test report shall confirm the intended operation at the supported PICC to PCD bit rates tested for Test procedure 1 and Test procedure 2.

The used test conditions shall be mentioned in the test report.

7.1.7 Load modulation reception for PICC to PCD bit rates of $f_c/8$, $f_c/4$ and $f_c/2$

7.1.7.1 Purpose

This test is used to verify that a PCD correctly detects the load modulation of a PICC which conforms to ISO/IEC 14443-2 for PICC to PCD bit rates of $f_c/8$, $f_c/4$ and $f_c/2$, if supported.

NOTE In future revisions of ISO/IEC 10373-6 a test similar to 7.1.6 can be defined also for these PICC to PCD bit rates.

7.1.7.2 Procedure

a) Tune the Reference PICC to 13,56 MHz as described in 5.5.2.4 and switch the jumper J1 to position 'c' and the jumper J2 to position 'b'.

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

c) Apply and adjust a DC voltage at CON2 to obtain a DC voltage at connector CON3 of V_{load} as defined in Table 3.

d) Increase the modulation signal amplitude at CON1 to produce responses until the PCD detects at least 10 of them consecutively.

e) Place the Reference PICC in the DUT position on the Test PCD assembly.

f) Adjust the Test PCD assembly to produce a field strength H which gives the same voltage at CON3 and note the corresponding field strength by reading the calibration coil voltage.

g) Measure the Reference PICC load modulation amplitude V_{LMA} as described in 7.2.1 and compare it with $V_{LMA,\ min,\ PCD}$ associated with the noted field strength. This measured V_{LMA} defines the PCD sensitivity criterion in order to compare with $V_{LMA,\ min,\ PCD}$ to perform these test measurements.

h) Repeat steps b) to g) for various positions within the operating volume for PICC to PCD bit rates of $f_c/8$, $f_c/4$ and $f_c/2$, if supported.

i) Repeat steps a) to h) with Reference PICC tuned to resonance frequency 15 MHz.

Any position in which the PCD sensitivity is above $V_{LMA,\ min,\ PCD}$ shall be considered out of the operating volume.

The test coverage may be expanded by using additional resonance frequencies below 13,56 MHz such as 12 MHz and 10 MHz.

The PCD sensitivity should be below $V_{LMA,\ min,\ PCD}$ to ensure good reception of the load modulation.

NOTE This test does not check that the PCD reception is independent of the phase of the load modulation. Consequently, it cannot guarantee the correct reception of any PICC compliant with ISO/IEC 14443-2.

7.1.7.3 Test report

The test report shall give the PCD load modulation sensitivity for the tested positions.

7.1.8 PCD EMD immunity test

7.1.8.1 Purpose

The purpose of this test is to determine whether the PCD is insensitive to any load modulation amplitude below $V_{E,\,PCD}$ for all supported PICC to PCD bit rates.

7.1.8.2 Procedure

a) Tune the Reference PICC to 13,56 MHz as described in 5.5.2.4 and switch the jumper J1 to position 'c'.

b) Switch the jumper J2 to position 'a' or to position 'b' according to the tested PICC to PCD bit rate and place the Reference PICC at a designated position in the PCD operating volume.

c) Apply and adjust a DC voltage at CON2 to obtain a DC voltage at CON3 of V_{load} as defined in Table 12.

d) Send the test pattern as shown in Figure 14. The test pattern is a valid standard frame including one single byte (01011101)b. The initial load modulation amplitude V_{EMD} of the test pattern shall be sufficiently low so that the PCD detects the PICC answer sent in step e).

e) Immediately after this test pattern, applying no gap, send the appropriate PICC answer to the PCD command with a PICC V_{LMA}, measured as defined in 7.2.1, of a value higher, e.g. twice, than the minimum value for the applied field strength H.

f) Increase V_{EMD} by adjusting the voltage at CON1 until the PCD does no longer detect the answer correctly. This may be determined by monitoring the next PCD command following the PICC answer; see Figure 14.

g) Place the Reference PICC into the DUT position on the Test PCD assembly.

h) Adjust the Test PCD assembly to produce a field strength H which gives the same voltage at CON3 and note the corresponding field strength by reading the calibration coil voltage.

i) Derive the current value of V_{EMD} on the Reference PICC by applying the power versus time measurement as described in 5.7.2.

j) Compare this measured V_{EMD} value with $V_{E,\,PCD}$.

k) Repeat steps b) to j) for other designated positions within the operating volume.

l) Repeat steps b) to k) for all supported PICC to PCD bit rates.

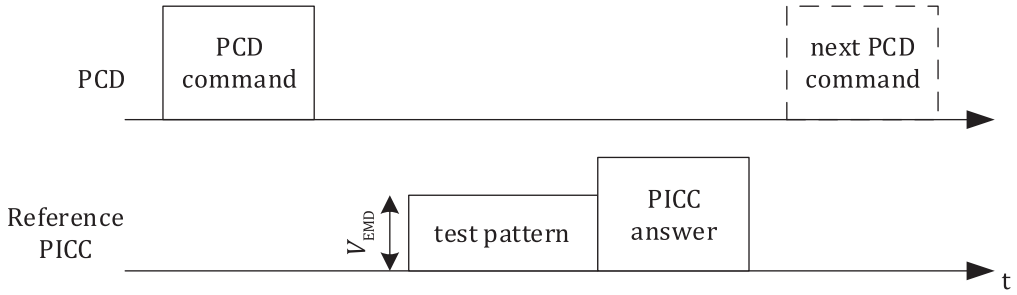


Figure 14 — PCD immunity test (common for Type A and Type B)

7.1.8.3 Test report

The test report shall state whether the PCD was insensitive to any load modulation amplitude below $V_{E,\,PCD}$ for all supported PICC to PCD bit rates.

7.2 PICC tests

7.2.1 PICC transmission

7.2.1.1 Purpose

The purpose of this test is to verify that the load modulation amplitude V_{LMA} of the PICC and the phase parameters $\emptyset_{LM,\,INTER}$ and $\emptyset_{LM,\,INTRA}$ of the PICC conform to ISO/IEC 14443-2 for all mandatory and supported optional PICC to PCD bit rates within the operating field range $[H_{min}, H_{max}]$.

7.2.1.2 Conditions

This test shall be done at ambient, minimum and maximum temperatures. If minimum and maximum temperatures are not provided in PICC test conditions, the PICC transmission test shall be performed only at ambient temperature.

7.2.1.3 Procedure

Step 1:

The load modulation test circuit of Figure 2 and the Test PCD assembly of Figure 3 defined for the PICC class (see ISO/IEC 14443-2:2020, 8.2.2.2) are used. If the PICC does not claim to meet the requirements of one particular PICC class as specified in ISO/IEC 14443-1, then use the Test PCD assembly 1.

Adjust the RF power delivered by the signal generator to the Test PCD antenna to the required field strength as measured by the calibration coil. Connect the output of the load modulation test circuit of Figure 2 to a signal acquiring device. The 10 Ω potentiometer P1 shall be trimmed to minimize the residual carrier. This signal shall be at least 40 dB lower than the signal obtained by shorting one sense coil. Additionally connect the output of the calibration coil (see Figure 1 and Figure 3) to the same signal acquiring device as the load modulation test circuit.

Step 2:

The PICC under test shall be placed in the DUT position (see Figure 3). If the PICC meets the requirements of one particular class as specified in ISO/IEC 14443-1, then its antenna shall be located within the zone defined for its PICC class centered in sense coil a. If the PICC does not claim to meet the requirements of one particular class as specified in ISO/IEC 14443-1, then its antenna shall be located within the external rectangle defined for “Class 1” centered in sense coil a. The RF drive into the Test PCD antenna shall be re-adjusted to the required field strength.

For each PICC to PCD bit rate supported by the PICC, the Test PCD shall put the PICC in the PROTOCOL state. An I-block shall be sent by the Test PCD to obtain a response from the PICC of at least the length defined in Table 13 for that PICC to PCD bit rate (TEST_COMMAND4 and TEST_RESPONSE4 fit this purpose). If this frame length is not supported by the PICC, its maximum supported frame length shall be used.

Table 13 — PICC transmission minimum frame length definition

PICC to PCD bit rate	Minimum frame length [bytes]
$f_c/128$	128
$f_c/64$	256
$f_c/32$	512

Table 13 *(continued)*

PICC to PCD bit rate	Minimum frame length [bytes]
$f_c/16$	1024
$f_c/8$	2048
$f_c/4$	4096
$f_c/2$	4096

NOTE These minimum frame lengths stem from the memory size of the signal acquiring device and can be increased in the future.

Display the whole PICC response of both, calibration coil and load modulation test circuit, including at least 200 carrier periods before the first and 20 carrier periods after the last modulation of the PICC on the signal acquiring device and store the sampled data of calibration coil and load modulation test circuit in separate files for analysis by computer programs as defined in Annex F and Annex N.

Load modulation amplitude computation:

Fourier transform with a Bartlett window exactly six subcarrier cycles of the sampled modulation waveform using suitable computer software (as the one given in Annex F). Use a discrete Fourier transformation with a scaling such that a pure sinusoidal signal results in its peak magnitude. In order to minimize transient effects, avoid analyzing a subcarrier cycle immediately following a non-modulating period or a phase shift of the subcarrier. The discrete Fourier transformation shall be done at the sidebands frequencies generated by the PICC under test, i.e. $f_c + f_s$ and $f_c - f_s$.

Amplitude and phase drift computation:

Use the PICC amplitude and phase drift analysis tool defined in Annex N.

This test procedure shall be repeated 5 times. The test apparatus shall generate the carrier frequency f_c as defined in ISO/IEC 14443-2:2020, 6.2:

1) once at the lower limit of the allowed range,

2) 3 times in the middle of the allowed range and

3) once at the upper limit of the allowed range.

Before each repetition the test apparatus shall perform a field reset of at least 5 ms.

7.2.1.4 Test report

The test report shall state:

1) Compliancy of the PICC load modulation amplitude V_{LMA} with the requirements defined in ISO/IEC 14443-2:

   a) For PICC to PCD bit rates of $f_c/128$, $f_c/64$, $f_c/32$ and $f_c/16$, if supported, the resulting peak amplitudes of the upper and lower sidebands at $f_c + f_s$ and $f_c - f_s$ shall be above $V_{LMA,\ min,\ PICC}$ as defined in ISO/IEC 14443-2:2020, 8.2.2.2.

   b) For PICC to PCD bit rates of $f_c/8$, $f_c/4$ and $f_c/2$, if supported, the average value resulting from the peak amplitudes of the upper and lower sidebands at $f_c + f_s$ and $f_c - f_s$ shall be above $V_{LMA,\ min,\ PICC}$ as defined in ISO/IEC 14443-2:2020, 8.2.2.2.

   c) For all PICC to PCD bit rates, the average value resulting from the peak amplitudes of the upper and lower sidebands at $f_c + f_s$ and $f_c - f_s$ shall be below $V_{LMA,\ max,\ PICC}$ as defined in ISO/IEC 14443-2:2020, 8.2.2.2.

2) Compliancy of the amplitude parameter $V_{|MS1-US|}$ of the PICC with the requirements defined ISO/IEC 14443-2:2020, 8.2.2.2 and phase parameters $\emptyset_{LM,\ INTRA}$ and $\emptyset_{LM,\ INTER}$ of the PICC with the

requirements defined ISO/IEC 14443-2:2020, 8.2.2.3 for all mandatory and supported optional PICC to PCD bit rates.

3) Based on the state information retrieved in N.9.1, compliancy of the following parameters of the PICC:

   a) For Type A PICC responses using a bit rate of $f_c/128$, the duration of an etu, the number of subcarrier periods per etu, whether 50 % of an etu are modulated with the subcarrier, and the start of communication shall comply with ISO/IEC 14443-2.

   b) For Type A PICC responses using a bit rate higher than $f_c/128$, if supported, the duration of an etu, the bit level encoding, and the start of communication shall comply with ISO/IEC 14443-2.

   c) For Type B PICC responses, the bit level encoding, TR1, SOF, character encoding, EGT and EOF shall comply with ISO/IEC 14443-3:2018.

Furthermore, the test report shall

1) give the measured peak amplitudes of the upper and lower sidebands at at $f_c + f_s$ and $f_c - f_s$ and the applied fields and modulations,

2) give the measured phase characteristics, and

3) confirm the intended operation at the mandatory PICC to PCD bit rate of $f_c/128$ and all supported optional PICC to PCD bit rates.

The used test conditions shall be mentioned in the test report.

7.2.2 PICC EMD level and low EMD time test

7.2.2.1 Purpose

The purpose of this test is to determine that the PICC does not generate an electromagnetic disturbance amplitude V_{EMD} higher than $V_{E,\,PICC}$ during $t_{E,\,PICC}$ with the exceptions defined in ISO/IEC 14443-2 for all supported PICC to PCD bit rates.

NOTE 1 The low EMD time $t_{E,\,PICC}$ is a function of FDT/TR0 as defined in ISO/IEC 14443-3:2018, 8.2.

NOTE 2 The EMD limit $V_{E,\,PICC}$ is a function of the field strength.

7.2.2.2 Noise requirements

In order to ensure a high dynamic range and sufficient sensitivity to EMD, the noise floor precondition test defined in 5.7.3 shall be performed before this test.

7.2.2.3 Test commands

The PICC EMD test shall be performed for ISO/IEC 14443-3 commands. Depending on the PICC application, additional higher layer commands shall be included in the test plan.

7.2.2.4 Procedure

This test shall be done at least applying H_{min} and $H_{max.}$ Using the Test PCD assembly, perform the following steps.

a) Adjust the RF power delivered by the signal generator to the Test PCD antenna to the required field strength as measured by the calibration coil.

b) Place the PICC under test into the DUT position. The RF drive into the Test PCD antenna shall be readjusted to the required field strength if necessary.

c) Reset the PICC by switching the RF field off and on; then if necessary send a transition of sequence commands to put the PICC into the Test Initial State, (see G.3.3.2.2 for PICC Type A and G.4.4.2.2 for PICC Type B).

d) Send the command to be tested.

e) Record the sense coil's signal for a time period of at least 200 µs before the start of PICC subcarrier generation. Additionally, record at least 50 µs after the first detected subcarrier in order to determine precisely the position of the PICC answer.

f) Determine the value of $t_{E,\ PICC}$ from the acquired signal: if the PCD modulation is present on the trace then measure the time between the last rising edge of PCD modulation and the start of PICC subcarrier generation and calculate $t_{E,\ PICC}$ with the formula given in ISO/IEC 14443-3:2018, 8.2; if the PCD modulation is not present on the trace then $t_{E,\ PICC}$ equals its maximum value defined in ISO/IEC 14443-3:2018, 8.2.

g) Compute the signal power at the frequencies $f_c + f_s$ and $f_c - f_s$ as a function of time as defined in 5.7.2.

h) Compute the maximum signal out of the two results obtained in g).

i) Determine the time t_{START} corresponding to half of the amplitude of the maximum signal obtained in h) during the rising edge of PICC transmission. Check if the amplitude of the maximum signal obtained in h) during the time period $[t_{START} - t_{E,\ PICC};\ t_{START} - 1/f_s]$ complies with the requirements defined in ISO/IEC 14443-2.

j) Repeat steps d) to i) for the next test command.

k) Repeat steps d) to j) for all supported PICC to PCD bit rates.

7.2.2.5 Test report

The test report shall state whether the PICC EMD level during $t_{E,\ PICC}$ complies with the requirements defined in ISO/IEC 14443-2 for all supported PICC to PCD bit rates.

Furthermore the test report shall give the measured maximum electromagnetic disturbance levels of the upper and lower sidebands at $f_c + f_s$ and $f_c - f_s$ during $t_{E,\ PICC}$. A graph showing EMD levels during $t_{E,\ PICC}$ should be incorporated in the report in case the test fails.

7.2.3 PICC reception

7.2.3.1 Purpose

The purpose of this test is to verify the ability of the PICC to receive the PCD commands.

7.2.3.2 General conditions

The test conditions shall be checked using the analysis tool defined in Annex E with the PICC under test in the DUT position. If at least one parameter is not within the tolerances defined in 4.3, the test conditions shall be readjusted.

The test PCD assembly may use a pre-equalization method and/or an impedance matching network different from those specified in A.2 to achieve some of the test conditions defined in Table 14, Table 15, Table 16 and Table 17.

For a frame size higher than 256 bytes, a frame with error correction as defined in ISO/IEC 14443-4 should be used.

7.2.3.3 PICC Type A for PCD to PICC bit rates of $f_c/128$, $f_c/64$, $f_c/32$ and $f_c/16$

7.2.3.3.1 Test conditions

The test conditions for a PCD to PICC bit rate of $f_c/128$ are specified in Table 14.

Table 14 — PICC Type A test conditions for a PCD to PICC bit rate of $f_c/128$

Test condition	t_1	t_2	t_3	t_4	PCD field envelope during 60 % of t_2[a]	Overshoot
1	$41/f_c$	$6/f_c$	$17/f_c$	$7/f_c$	0,5 %	0 % (100 % of $H_{INITIAL}$)
2	$27{,}5/f_c$	$25{,}5/f_c$	$10/f_c$	$6/f_c$	0,5 %	10 % (110 % of $H_{INITIAL}$)
3	$41/f_c$	$13/f_c$	$2/f_c$		0,5 %	10 % (110 % of $H_{INITIAL}$)
4	$40/f_c$	$7/f_c$	$16/f_c$	$6/f_c$	4 %	0 % (100 % of $H_{INITIAL}$)
5	$28{,}5/f_c$	$26/f_c$	$10/f_c$	$6/f_c$	4 %	10 % (110 % of $H_{INITIAL}$)
[a] During the remaining 40 % of t_2, the PCD field envelope shall be at least the defined value.						

All of these test conditions shall be tested at least using H_{min} and H_{max}.

The test conditions for PCD to PICC bit rates of $f_c/64$, $f_c/32$ and $f_c/16$ are specified for each bit rate in Table 15.

Table 15 — PICC Type A test conditions for PCD to PICC bit rates of $f_c/64$, $f_c/32$ and $f_c/16$

PCD to PICC bit rate	Test condition	t_1	t_5	t_6	a	h_{ovs}
$f_c/64$	1	$20/f_c$	$13/f_c$	$12/f_c$	0,2	0 %
	2	$16/f_c$	$15/f_c$	$10/f_c$	0	6,4 %
	3	$20/f_c$	$14/f_c$	$2/f_c$	0	10,1 %
$f_c/32$	1	$10/f_c$	$6/f_c$	$10/f_c$	0,4	0 %
	2	$8/f_c$	$7/f_c$	$9/f_c$	0	6,1 %
	3	$10/f_c$	$6/f_c$	$2/f_c$	0	9,9 %
$f_c/16$	1	$5/f_c$	$2{,}5/f_c$	$6/f_c$	0,6	0 %
	2	$4/f_c$	$3/f_c$	$5{,}5/f_c$	0,2	4,8 %
	3	$5/f_c$	$2{,}5/f_c$	$2/f_c$	0,2	7,3 %

All of these test conditions shall be tested at least using H_{min} and H_{max}.

7.2.3.3.2 Procedure

Under all of the test conditions defined in Table 14 the PICC shall answer to a REQA with ATQA.

Under test conditions 2 and 3 defined in Table 14 the PICC shall answer to REQA with ATQA and to WUPA with ATQA respecting the Frame Delay Time as defined in ISO/IEC 14443-3:2018, 6.2.1.1, where the Frame Delay Time shall be determined by the method defined in Annex P.

For each optional PCD to PICC bit rate supported by the PICC, the PICC shall operate under all of the conditions defined in Table 15 after selection of that optional PCD to PICC bit rate. The PICC shall respond correctly to an I-block transmitted at that optional PCD to PICC bit rate.

For each optional PCD to PICC bit rate in combination with a PICC to PCD bit rate of $f_c/128$, the PICC shall operate under each of the test conditions 2 and 3 defined in Table 15 after selection of that optional PCD to PICC bit rate. The PICC shall respond to an I-block with (0)b as last bit and an I-block with (1)b as last bit transmitted at that optional PCD to PICC bit rate respecting the Frame Delay Time defined in in ISO/IEC 14443-3:2018, 6.2.1.1, where the Frame Delay Time shall be determined by the method defined in Annex P.

7.2.3.4 PICC Type B for PCD to PICC bit rates of $f_c/128$, $f_c/64$, $f_c/32$ and $f_c/16$

7.2.3.4.1 Test conditions

The test conditions for each bit rate are defined in Table 16.

Table 16 — PICC Type B test conditions for PCD to PICC bit rates of $f_c/128$, $f_c/64$, $f_c/32$ and $f_c/16$

PCD to PICC bit rate	Test condition	t_f	t_r	h_f	h_r	m	b
$f_c/128$	1	$17/f_c$	$17/f_c$	0 %	0 %	7 %	0,869
	2	$2/f_c$	$11/f_c$	1,4 %	1 %	7 %	0,869
	3	$11/f_c$	$2/f_c$	1 %	1,4 %	7 %	0,869
	4	$17/f_c$	$17/f_c$	0 %	0 %	15 %	0,739
	5	$2/f_c$	$11/f_c$	2,7 %	1,9 %	15 %	0,739
	6	$11/f_c$	$2/f_c$	1,9 %	2,7 %	15 %	0,739
$f_c/64$	1	$14/f_c$	$14/f_c$	0 %	0 %	7 %	0,869
	2	$2/f_c$	$9/f_c$	1,3 %	1 %	7 %	0,869
	3	$9/f_c$	$2/f_c$	1 %	1,3 %	7 %	0,869
	4	$14/f_c$	$14/f_c$	0 %	0 %	15 %	0,739
	5	$2/f_c$	$9/f_c$	2,7 %	1,9 %	15 %	0,739
	6	$9/f_c$	$2/f_c$	1,9 %	2,7 %	15 %	0,739
$f_c/32$	1	$11/f_c$	$11/f_c$	0 %	0 %	7 %	0,869
	2	$2/f_c$	$7,5/f_c$	1,3 %	0,9 %	7 %	0,869
	3	$7,5/f_c$	$2/f_c$	0,9 %	1,3 %	7 %	0,869
	4	$11/f_c$	$11/f_c$	0 %	0 %	15 %	0,739
	5	$2/f_c$	$7,5/f_c$	2,6 %	1,9 %	15 %	0,739
	6	$7,5/f_c$	$2/f_c$	1,9 %	2,6 %	15 %	0,739
$f_c/16$	1	$8/f_c$	$8/f_c$	0 %	0 %	7 %	0,869
	2	$2/f_c$	$6/f_c$	1,3 %	0,9 %	7 %	0,869
	3	$6/f_c$	$2/f_c$	0,9 %	1,3 %	7 %	0,869
	4	$8/f_c$	$8/f_c$	0 %	0 %	15 %	0,739
	5	$2/f_c$	$6/f_c$	2,5 %	1,8 %	15 %	0,739
	6	$6/f_c$	$2/f_c$	1,8 %	2,5 %	15 %	0,739

All of these test conditions shall be tested at least using H_{min} and H_{max}.

7.2.3.4.2 Procedure

Under all of the test conditions defined in Table 16, the PICC operating at a bit rate of $f_c/128$ shall answer to a REQB with ATQB.

For each optional PCD to PICC bit rate supported by the PICC, the PICC shall operate under each of the test conditions defined in Table 16 after selection of that optional PCD to PICC bit rate. This PICC shall respond correctly to an I-block transmitted at that optional PCD to PICC bit rate.

7.2.3.5 PICC Type A or Type B for PCD to PICC bit rates of $f_c/8$, $f_c/4$ and $f_c/2$

7.2.3.5.1 Test conditions

Six test conditions are defined for each bit rate in Table 17.

Table 17 — PICC test conditions for PCD to PICC bit rates of $f_c/8$, $f_c/4$ and $f_c/2$

PCD to PICC bit rate	Test condition	t_f	t_r	h_f	h_r	m	b
$f_c/8$	1	$6/f_c$	$6/f_c$	0 %	0 %	8 %	0,852
	2	$2/f_c$	$5/f_c$	1,4 %	1 %	8 %	0,852
	3	$5/f_c$	$2/f_c$	1 %	1,4 %	8 %	0,852
	4	$6/f_c$	$6/f_c$	0 %	0 %	21 %	0,653
	5	$2/f_c$	$5/f_c$	3,2 %	2,2 %	21 %	0,653
	6	$5/f_c$	$2/f_c$	2,2 %	3,2 %	21 %	0,653
$f_c/4$	1	$4/f_c$	$4/f_c$	0 %	0 %	8 %	0,852
	2	$2/f_c$	$4/f_c$	1,2 %	0,8 %	8 %	0,852
	3	$4/f_c$	$2/f_c$	0,8 %	1,2 %	8 %	0,852
	4	$4/f_c$	$4/f_c$	0 %	0 %	21 %	0,653
	5	$2/f_c$	$4/f_c$	2,9 %	1,9 %	21 %	0,653
	6	$4/f_c$	$2/f_c$	1,9 %	2,9 %	21 %	0,653
$f_c/2$	1	$3/f_c$	$3/f_c$	0 %	0 %	8 %	0,852
	2	$2/f_c$	$3/f_c$	1,1 %	0,8 %	8 %	0,852
	3	$3/f_c$	$2/f_c$	0,8 %	1,1 %	8 %	0,852
	4	$3/f_c$	$3/f_c$	0 %	0 %	21 %	0,653
	5	$2/f_c$	$3/f_c$	2,5 %	1,9 %	21 %	0,653
	6	$3/f_c$	$2/f_c$	1,9 %	2,5 %	21 %	0,653

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

7.2.3.5.2 Procedure

For each optional PCD to PICC bit rate supported by the PICC, the PICC shall operate under each of the test conditions defined in Table 17 after selection of that optional PCD to PICC bit rate. This PICC shall respond correctly to an I-block transmitted at that optional PCD to PICC bit rate.

7.2.3.6 PICC Type A or Type B for PCD to PICC bit rates of $3f_c/4$, f_c, $3f_c/2$ and $2f_c$

See K.2.2.1.

7.2.3.7 Test report

The test report shall confirm the intended operation at the mandatory $f_c/128$ bit rate. For PICCs supporting one or more of the optional PCD to PICC bit rates, the test report shall confirm the intended operation at the supported bit rates.

The used test conditions shall be mentioned in the test report.

7.2.4 PICC resonance frequency

7.2.4.1 Purpose

The test may be used to measure the resonance frequency of a PICC.

When two or more PICCs are placed in the same PCD energizing field, the resonance frequency of each PICC decreases.

Care should be taken in designing each PICC resonance frequency.

WARNING — The resonance frequency may depend on the field strength used during the measurement.

7.2.4.2 Procedure

The resonance frequency of a PICC is measured by using an impedance analyzer or a network analyzer or a LCR-meter connected to a calibration coil. The PICC should be placed on the calibration coil at a distance of 10 mm, with the axes of the two coils being congruent. The resonance frequency is that frequency at which the resistive part of the measured complex impedance is at maximum.

7.2.4.3 Test report

When applied the test report shall give the PICC resonance frequency and the measurement conditions.

7.2.5 PICC maximum loading effect

7.2.5.1 Purpose

This test is used to measure the PICC loading effect.

7.2.5.2 Procedure

Depending on the claimed PICC class, select:

1) the relevant H_{min} as defined in ISO/IEC 14443-2:2020, Table 2;

2) the relevant Reference PICC as defined in Table 12 and its voltage V_{load};

3) the relevant Test PCD assembly as defined in Table 12.

If the PICC does not claim any particular PICC class as specified in ISO/IEC 14443-1, then PICC parameters, test apparatus and circuits in accordance with ISO/IEC 14443-2:2020, 6.3 shall be used for this test.

The PICC loading effect at H_{min} shall be measured using the Test PCD assembly. It shall be less than the loading effect of the selected Reference PICC tuned to 13,56 MHz and calibrated to obtain V_{load} at CON3 at H_{min}.

The default procedure of this substitution method is as follows.

a) Tune the selected Reference PICC to 13,56 MHz as described in 5.5.2.4.

b) Calibrate the Test PCD assembly to produce the H_{min} operating condition on the calibration coil.

c) Place the Reference PICC into the DUT position on the Test PCD assembly. Switch the jumper J1 to position 'b' and adjust R2 to obtain a DC voltage of V_{load} measured at CON3. Alternatively, jumper J1 may be set to position 'c' and the applied voltage on CON2 is adjusted to obtain a DC voltage of V_{load} at CON3. In both cases, the operating field condition shall be verified by monitoring the voltage on the calibration coil and adjusted if necessary.

   WARNING — R2 value should be between R2min and R2max as defined in Table 12. Check this range at least once before using the alternative method.

d) Remove the Reference PICC.

   WARNING —The unloaded field strength when Reference PICC 1 using a V_{load} of 6 V is removed should be between 1,70 A/m and 1,90 A/m. The unloaded field strength when Reference PICC 1 using a V_{load} of 4,5 V is removed should be between 1,65 A/m and 1,85 A/m. The unloaded field strength when Reference PICC 2 or Reference PICC 3 is removed should be between 1,58 A/m and 1,70 A/m.

e) Place the PICC under test into the DUT position on the Test PCD assembly.

f) Measure the field strength monitored by the calibration coil. This field strength shall be greater than H_{min}.

If PICC Class 1 parameters were used and if the field strength measured in step f) was not greater than H_{min}, then the following extended procedure may be used to measure the PICC loading effect:

1) repeat steps b) to f) using Reference PICC 1 configured for a V_{load} of 6 V instead of the voltage defined in Table 12;

NOTE 1 The warning about R2 value is not applicable in this case.

2) repeat the PICC transmission test as defined in 7.2.1 with a field strength of H_{min} – 0,2 A/m (rms) instead of H_{min} as defined in ISO/IEC 14443-2:2020 Table 2, using the PICC V_{LMA} limits defined for H_{min}, as defined in ISO/IEC 14443-2:2020, Table 22;

3) repeat the PICC EMD level and low EMD time test as defined in 7.2.2 with a field strength of H_{min} – 0,2 A/m (rms) instead of H_{min} as defined in ISO/IEC 14443-2:2020, Table 2 using the $V_{E, PICC}$ limit defined for H_{min} as defined in ISO/IEC 14443-2:2020, 10.2;

4) repeat the PICC reception test as defined in 7.2.3 with a field strength of H_{min} – 0,2 A/m (rms) instead of H_{min} as defined in ISO/IEC 14443-2:2020, Table 2, using for PICC Type B the modulation index m defined for H_{min} as defined in ISO/IEC 14443-2:2020, 9.1.2;

5) repeat the PICC operating field strength test as defined in 7.2.6 with a field strength of H_{min} – 0,2 A/m (rms) instead of H_{min} as defined in ISO/IEC 14443-2:2020, Table 2.

NOTE 2 This extended procedure checks that a slightly higher PICC loading effect is compensated by a slightly lower PICC minimum operating field strength.

7.2.5.3 Test report

If the extended procedure has not been used, the test result is only PASS if the field strength measured in step f) is greater than H_{min}, otherwise the test result is FAIL.

If the extended procedure has been used, the test result is only PASS if the field strength measured in step f) is greater than H_{min} (using Reference PICC 1 configured for a DC voltage V_{load} of 6 V), and if

— the PICC transmission test as defined in 7.2.1,

— the PICC EMD level and low EMD time test as defined in 7.2.2,

— the PICC reception test as defined in 7.2.3, and

— the PICC operating field strength test as defined in 7.2.6

are PASS with a field strength of H_{min} – 0,2 A/m (rms) instead of H_{min}, otherwise the test result is FAIL.

The test report shall give the value of the measured field strength and the V_{load} value used.

7.2.6 PICC operating field strength test

7.2.6.1 Purpose

This test verifies that the PICC operates as intended between H_{min} and H_{max} for all supported PCD to PICC bit rates.

7.2.6.2 Conditions

This test shall be done at least at ambient, minimum and maximum temperatures applying H_{min} and H_{max}. If minimum and maximum temperatures are not provided in PICC test conditions, PICC operating field strength test shall be performed only at ambient temperature.

When the test condition is both H_{max} and maximum temperature, the PICC should be put in horizontal position. The impact of the ventilation (if any) should be minimized during the test in order to limit the PICC's heat dissipation. The test shall be preceded with a one-minute PICC exposition to a field strength of H_{max} without any field shut-off so that the PICC chip has reached its maximum temperature when the test starts. The air temperature value during test execution shall remain within ±3 °C of the nominal value.

7.2.6.3 Procedure

For each PCD to PICC bit rate supported by the PICC, the following command sequence shall be used for this procedure using the test conditions defined in 7.2.6.2.

For Type A:

a) REQA command.

b) Repeat ANTICOLLISION and SELECT commands until UID is complete.

c) RATS command.

d) PPS command.

e) S(PARAMETERS) commands.

f) TEST_COMMAND_SEQUENCE1.

NOTE 1 The PICC may not support PPS or S(PARAMETERS) commands.

For Type B:

a) REQB command

b) ATTRIB command

c) S(PARAMETERS) commands

d) TEST_COMMAND_SEQUENCE1

NOTE 2 The PICC may not support S(PARAMETERS) commands.

7.2.6.4 Test report

The test report shall confirm the intended operation at the mandatory $f_c/128$ bit rate. For PICCs supporting one or more of the optional PCD to PICC bit rates the test report shall confirm the intended operation at the supported PCD to PICC bit rates.

The used test conditions shall be mentioned in the test report.

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

Annex K shall apply.

7.4 PXD tests

PCD and PICC tests shall be applied as follows:

1) when the PXD is in PCD Mode, tests defined in 7.1 and optionally 7.3 shall be applied;

2) when the PXD is in PICC Mode, tests defined in 7.2 and optionally 7.3 shall be applied.

NOTE In automatic mode alternation, the PXD may be forced into the required mode.

8 Test of ISO/IEC 14443-3 and ISO/IEC 14443-4 parameters

8.1 PCD tests

8.1.1 PCD EMD recovery test

8.1.1.1 Purpose

The purpose of this test is to determine whether the PCD is disturbed by a test pattern sent $t_{E,\ PCD}$ before the PICC answer for all supported PICC to PCD bit rates.

8.1.1.2 Procedure

a) The UT performs the protocol activation procedure according to H.1.9.2 for Type A or H.1.9.3 for Type B.

b) The UT sends the SEND_UT_APDU(UT_TEST_COMMAND1) to the PCD.

c) The LT waits until the PCD sends an I-block to the LT with the INF field containing the UT_TEST_COMMAND1.

d) The LT sends in sequence, as illustrated in Figure 15 using the $t_{E,\ PCD}$ associated with minimum FDT/TR0,

NOTE 1 The low EMD time $t_{E,\ PCD}$ is a function of FDT/TR0 as defined in ISO/IEC 14443-3:2018, 8.2.

1) a test pattern, which starts sending the two data bits b1 = (0)b followed by b2 = (1)b in a valid way to the PCD, but interrupts immediately after the second bit sent, as illustrated in Figure 16 for Type A and Figure 17 for Type B,

NOTE 2 Depending on the FDT/TR0, the test pattern may start before the end of the PCD command.

2) a period with no load modulation for a duration of $t_{E,\ PCD}$,

3) the appropriate answer to the PCD command.

e) Check if the PCD behaves in the same way as if there was no test pattern. This may be determined by monitoring the next PCD command following the PICC answer, see Figure 15.

f) Repeat steps d) and e) 10 times.

g) Repeat steps d) to f) replacing minimum FDT/TR0 with maximum FDT/TR0.

h) Repeat steps a) to g) for all supported PICC to PCD bit rates.

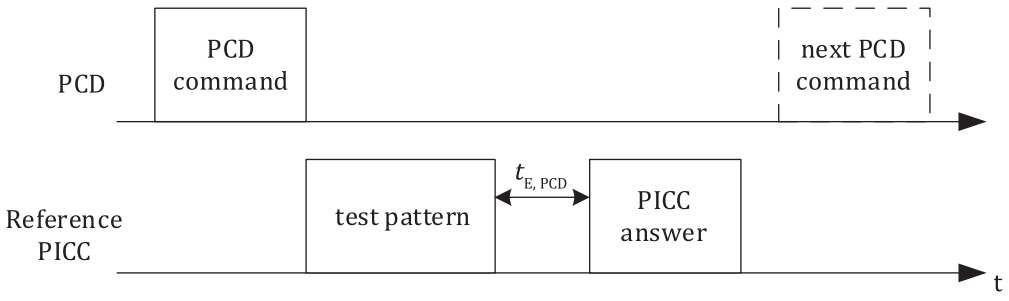


Figure 15 — EMD recovery test sequence (common for Type A and Type B)

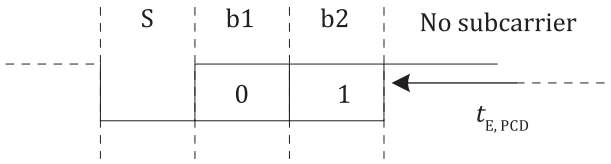


Figure 16 — Test pattern for the EMD recovery test (Type A)

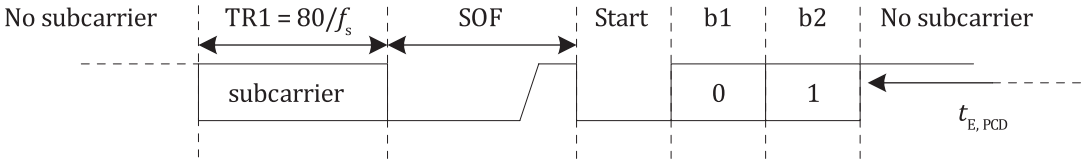


Figure 17 — Test pattern for the EMD recovery test (Type B

8.1.1.3 Test report

The test report shall report whether the PCD was not disturbed by the test pattern sent $t_{E, PCD}$ before the PICC answer (or was able to recover from the test pattern) for all supported PICC to PCD bit rates.

8.1.2 Additional PCD tests

Additional PCD tests of ISO/IEC 14443-3 and ISO/IEC 14443-4 parameters shall be as defined in Annex H, Annex I and Annex L.

8.2 PICC tests

PICC tests of ISO/IEC 14443-3 and ISO/IEC 14443-4 parameters shall be as defined in Annex G and Annex L.

8.3 PXD tests

8.3.1 PCD and PICC Modes

PCD and PICC tests shall be applied as follows:

1) when the PXD is in PCD Mode, tests defined in 8.1 shall be applied;

2) when the PXD is in PICC Mode, tests defined in 8.2 shall be applied.

NOTE In automatic mode alternation, the PXD may be forced into the required mode.

8.3.2 Automatic mode alternation

8.3.2.1 General

The tests defined in this subclause apply only to PXD supporting automatic mode alternation.

One cycle is defined as the duration between two consecutive beginnings of PCD Mode (RF field on).

8.3.2.2 PCD Mode and PICC Mode alternation cycle

8.3.2.2.1 Purpose

This test checks that

1) each cycle does not last longer than t_{cyc},
2) in each cycle, the PICC Mode lasts longer than PCD Mode, and
3) the PICC Mode duration varies randomly and differs by at least t_{diff}.

8.3.2.2.2 Conditions

The PXD should not be in close proximity to another PXD, PCD or PICC.

8.3.2.2.3 Procedure

The RF field of the PXD shall be monitored and evaluated for at least 10 consecutive cycles.

a) Ensure that the PXD is in automatic Mode alternation.
b) Measure all RF field on and RF field off durations.

8.3.2.2.4 Test report

The test result is PASS if all the following conditions are met:

1) no cycle lasts more than t_{cyc};
2) for each t_{cyc} the PICC Mode duration (RF field off) is longer than the PCD Mode duration (RF field on);
3) the PICC Mode durations vary;
4) the minimum and maximum PICC Mode durations differ by at least t_{diff}.

Otherwise the test result is FAIL.

NOTE 1 The appreciation of the randomness of the PICC Mode duration can be done with common statistical methods.

NOTE 2 Due to statistical reasons the test result can be FAIL and the test can be repeated.

8.3.2.3 PCD Mode

8.3.2.3.1 Polling

8.3.2.3.1.1 Purpose

This test checks that the PXD in automatic mode alternation polls for Type A and Type B PICCs as defined in ISO/IEC 14443-3:2018, 5.3 in each cycle of PCD Mode.

8.3.2.3.1.2 Conditions

The PXD should not be in close proximity to another PXD, PCD or PICC.

8.3.2.3.1.3 Procedure

All modulations during PCD Mode shall be monitored, and the timings between the Request commands shall be measured for at least 10 consecutive cycles.

a) Ensure that the PXD is in automatic alternation mode.

b) Monitor all Request commands during PCD Mode.

c) Measure timings before and between each command.

8.3.2.3.1.4 Test report

The test result is PASS if all the following conditions are met in each cycle:

1) at least one REQA/WUPA command is sent by the PXD;

2) at least one REQB/WUPB command is sent by the PXD;

3) the duration of unmodulated field before at least one of the REQA/WUPA commands is more than 5 ms;

4) the duration of unmodulated field before at least one of the REQB/WUPB commands is more than 5 ms.

Otherwise the test result is FAIL.

8.3.2.3.2 End of PCD Mode

8.3.2.3.2.1 Purpose

This test checks that the PXD in automatic mode alternation leaves the PCD Mode after processing of a PICC, and resumes its automatic mode alternation with the PICC Mode first.

8.3.2.3.2.2 Conditions

The PCD-test-apparatus shall be used.

8.3.2.3.2.3 Procedure

Perform the following steps:

a) Place the PCD-test-apparatus in the operating volume of the PXD.

b) Send responses to all anticollision commands sent by the PXD until the PCD-test-apparatus is in ACTIVE or PROTOCOL state.

c) Do not answer any further PXD commands.

8.3.2.3.2.4 Test report

The test result is PASS if the PXD resumes its automatic mode alternation, possibly after application of the error handling or PICC presence check rules, with the PICC Mode first, otherwise the test result is FAIL.

8.3.2.4 PICC Mode

8.3.2.4.1 Reaction to polling

8.3.2.4.1.1 Purpose

This test checks that the PXD in automatic mode alternation responds to Type A or Type B Request commands as defined in ISO/IEC 14443-3:2018, 5.3 in each cycle of PICC Mode.

8.3.2.4.1.2 Conditions

The PICC-test-apparatus shall be used.

8.3.2.4.1.3 Procedure 1

Perform the following steps:

a) Switch the PICC-test-apparatus RF operating field off.

b) Place the PXD into the test position of the PICC-test-apparatus and ensure that the PXD is in automatic alternation mode.

c) Switch the PICC-test-apparatus RF operating field on while the PXD RF field is on.

d) Send a REQA command 5 ms after the start of PICC Mode (PXD RF field off).

e) Send a REQB command 5 ms after the end the REQA command.

f) Record the presence and the content of the PXD response.

8.3.2.4.1.4 Procedure 2

Perform the following steps:

a) Switch the PICC-test-apparatus RF operating field off.

b) Place the PXD into the test position of the PICC-test-apparatus and ensure that the PXD is in automatic alternation mode.

c) Switch the PICC-test-apparatus RF operating field on while the PXD RF field is on.

d) Send a REQB command 5 ms after the start of PICC Mode (PXD RF field off).

e) Send a REQA command 5 ms after the end the REQB command.

f) Record the presence and the content of the PXD response.

8.3.2.4.1.5 Test report

The test result is PASS if the PXD response is one of the following:

1) ATQA in each of the two procedures (Type A PICC Mode);

2) ATQB in each of the two procedures (Type B PICC Mode);

3) ATQA in Procedure 1 and ATQB in Procedure 2 (Type A and Type B PICC Mode).

Otherwise the test result is FAIL.

8.3.2.4.2 PICC Mode duration and exit conditions

8.3.2.4.2.1 Purpose

This test checks that, after reception of a valid REQA/WUPA command or REQB/WUPB command, the PXD in automatic mode alternation does not go in PCD Mode before a POWER-OFF state.

8.3.2.4.2.2 Conditions

The PICC-test-apparatus shall be used.

8.3.2.4.2.3 Procedure

Perform the following steps:

a) Switch the PICC-test-apparatus RF operating field on.

b) Ensure that the PXD is in automatic alternation mode and place the PXD into the test position of the PICC-test-apparatus.

c) Send a REQA command at least 5 ms after the start of PICC Mode (PXD RF field off).

d) Send a REQB command 5 ms after the end of the REQA command if there is no answer to the REQA command.

e) Keep the PICC-test-apparatus RF field on for more than 2 s.

f) Continue with anticollision commands to put the PXD in ACTIVE or PROTOCOL state and check the PXD responses.

g) Send a HALT or S(DESELECT) command to put the PXD in HALT state.

h) Keep the PICC-test-apparatus RF field on for more than 2 s.

i) Send a Request command of the communication signal interface which was answered in step c) or d) and check there is no PXD response.

j) Send a Wake-up command of the communication signal interface which was answered in step c) or d) and check there is a PXD response.

k) Switch the PICC-test-apparatus RF operating field off.

l) Check that the automatic alternation resumes in less than 1 s by monitoring the PXD RF field.

8.3.2.4.2.4 Test report

The test result is PASS if all steps of the procedure succeed, otherwise the test result is FAIL.

Annex A
(normative)

Test PCD antennas

A.1 Test PCD antenna 1 layout including impedance matching network

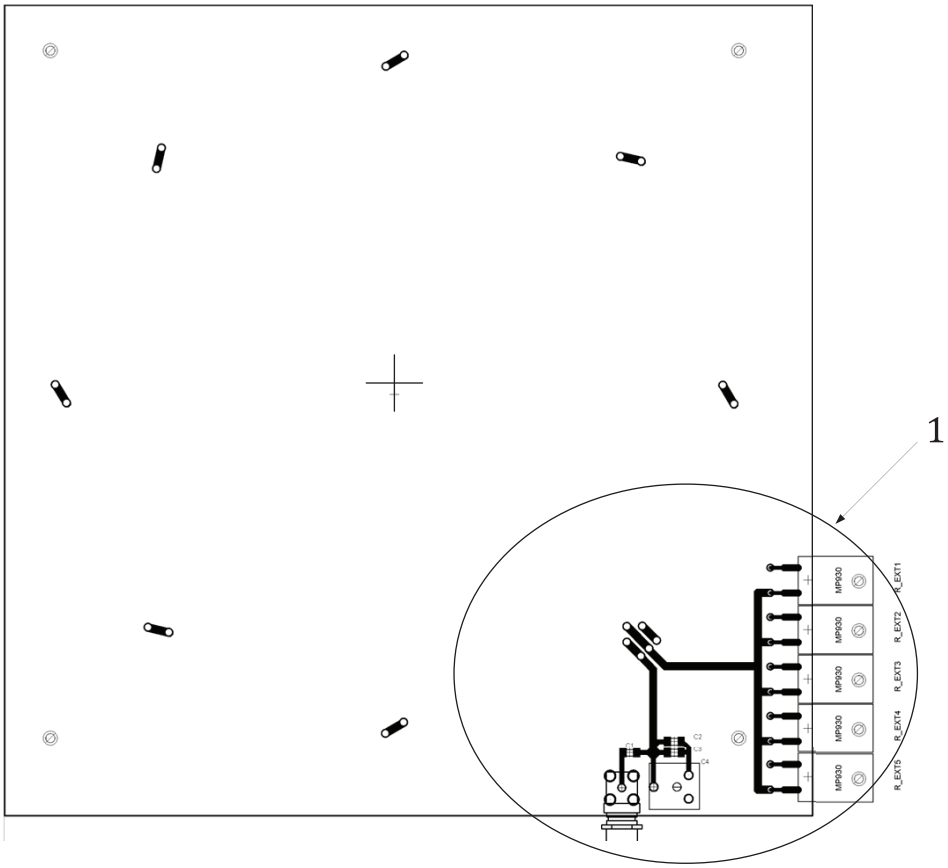
Figure A.1, Figure A.2, Figure A.3, and Figure A.4 illustrate Test PCD antenna 1 layout.

The antenna coil track width is 1,8 mm (except for through-plated holes).

Starting from the impedance matching network, there are crossovers every 45°.

Printed circuit board (PCB): FR4 material, thickness 1,6 mm, double sided with 35 µm copper.

NOTE The layout of the impedance matching network is informative.



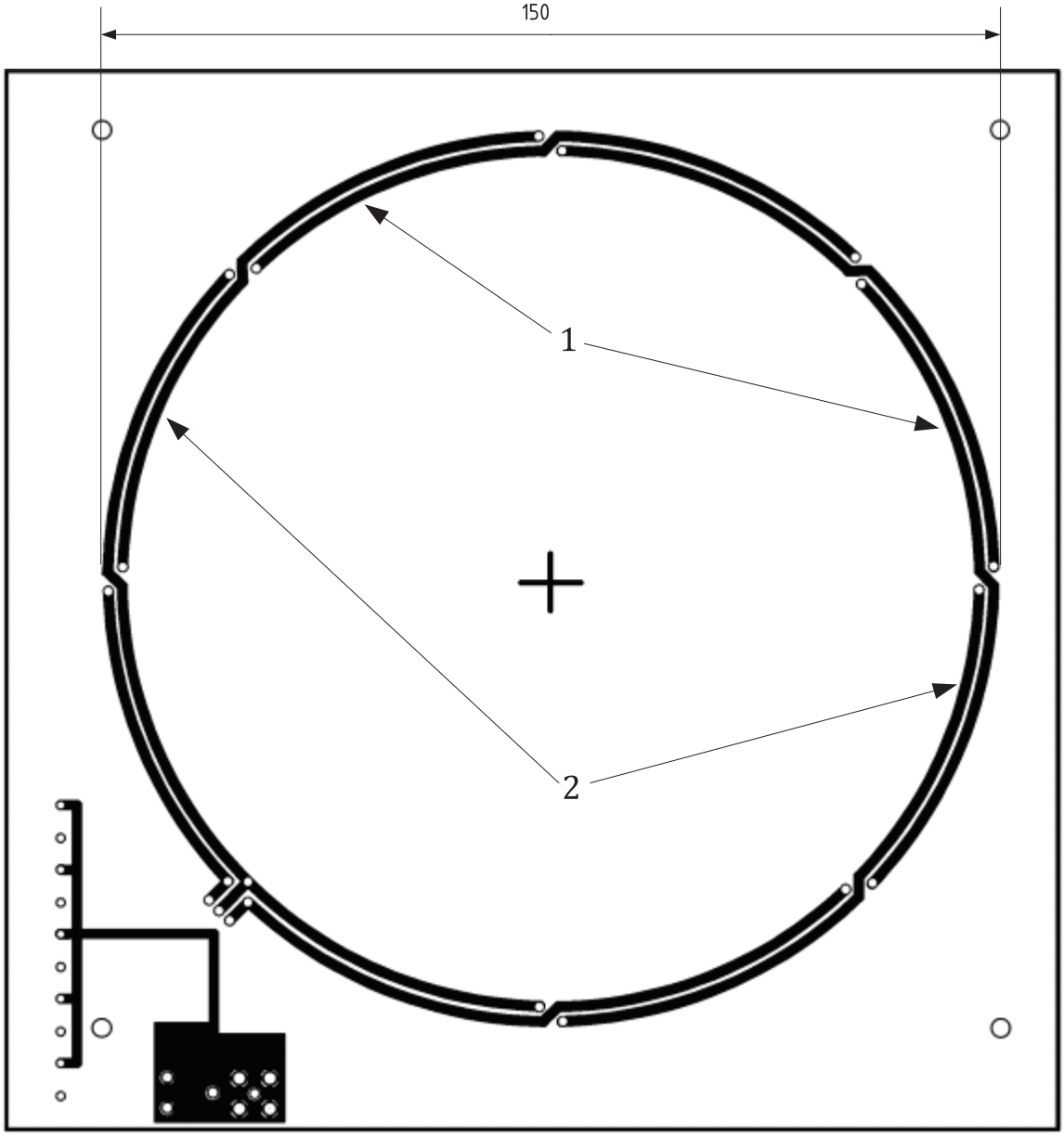
Key

1 impedance matching network

NOTE Drawing is not to scale.

Figure A.1 — Test PCD antenna 1 layout including impedance matching network for a bit rate of $f_c/128$ (view from front)

Dimensions in millimetres



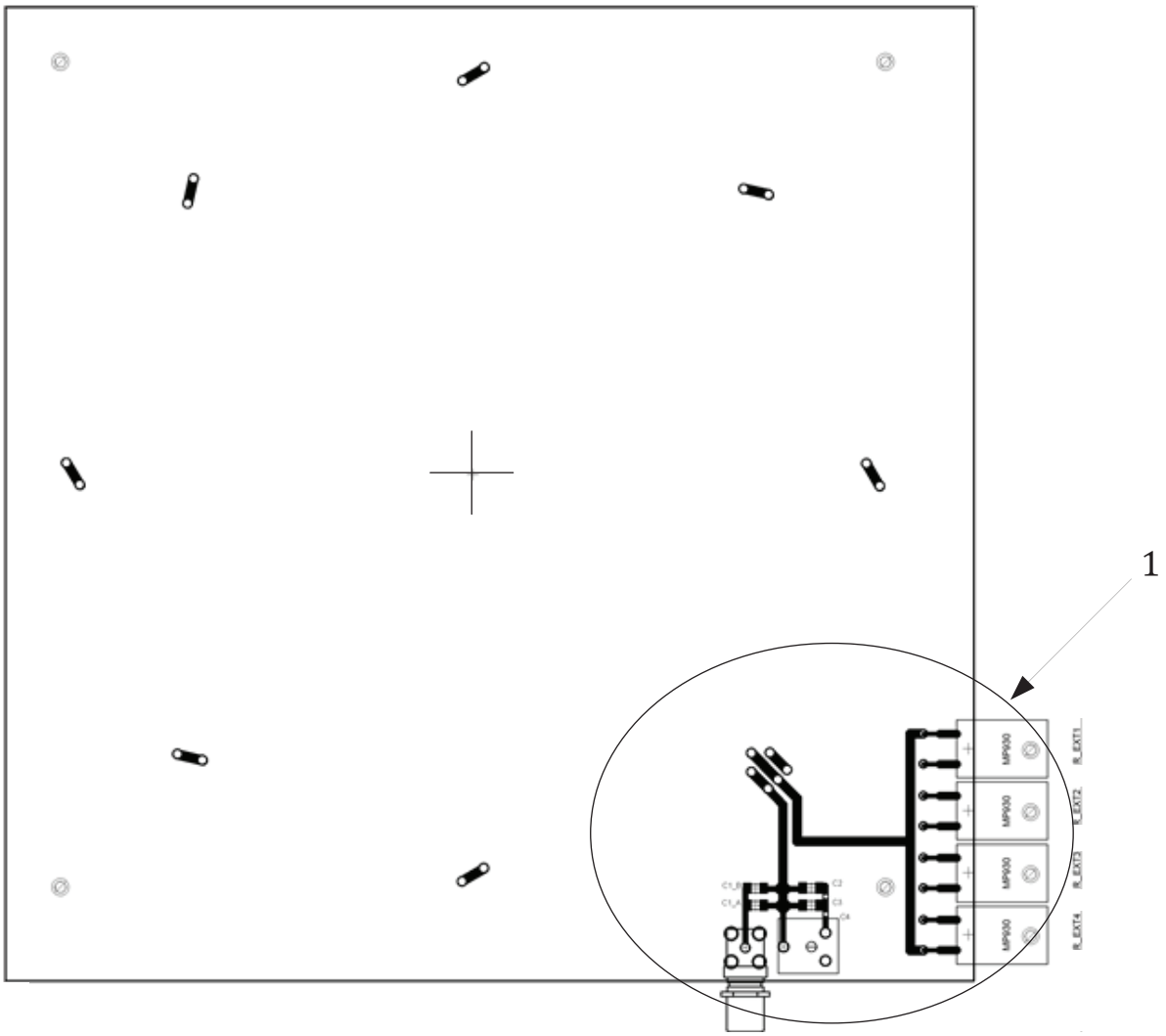
Key

1 antenna coil

2 ground compensation coil

NOTE Drawing is not to scale.

Figure A.2 — Test PCD antenna 1 layout including impedance matching network for a bit rate of $f_c/128$ (view from back)



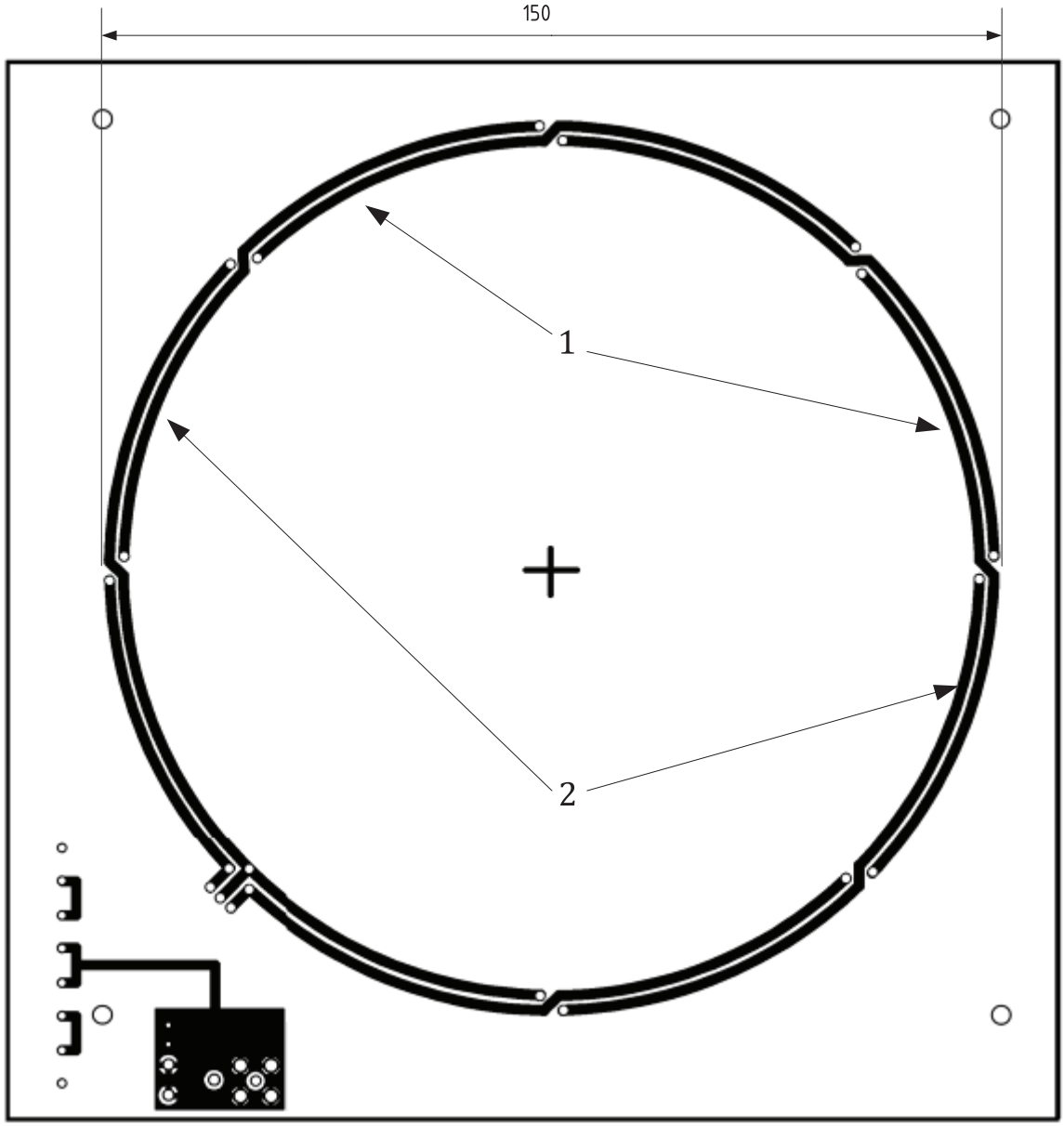
Key

1 impedance matching network

NOTE Drawing is not to scale.

Figure A.3 — Test PCD antenna 1 layout including impedance matching network for bit rates higher than $f_c/128$ (view from front)

Dimensions in millimetres



Key

1 antenna coil

2 ground compensation coil

NOTE Drawing is not to scale.

Figure A.4 — Test PCD antenna 1 layout including impedance matching network for bit rates higher than $f_c/128$ (view from back)

A.2 Impedance matching network

A.2.1 General

The antenna impedance (R_{ant}, L_{ant}) is adapted to the signal generator output impedance (Z = 50 Ω) by a matching circuit (see A.2.2 and A.2.3). The capacitors C1, C1a, C1b, C2 and C3 have fixed values. The input impedance phase can be adjusted with the variable capacitor C4.

The Test PCD assembly as defined in 5.4 and in this Annex is intended to be used for time limited measurements, to avoid any overheating of the individual components. If the test is run continuously, heat dissipation shall be improved. Care shall be taken to keep maximum voltages and maximum heat dissipation within the specified limits of the individual components.

Two impedance matching networks are described: the impedance matching network for a bit rate of $f_c/128$ and the impedance matching network for bit rates higher than $f_c/128$; see Table A.1.

Table A.1 — Impedance matching network use cases

Type of impedance matching network	Magnetic field capability	PCD to PICC bit rate capability
Impedance matching network for a bit rate of $f_c/128$, see A.2.2	Up to 12 A/m (rms)	Only $f_c/128$
Impedance matching network for bit rates higher than $f_c/128$, see A.2.3	Up to 7,5 A/m (rms)	All bit rates

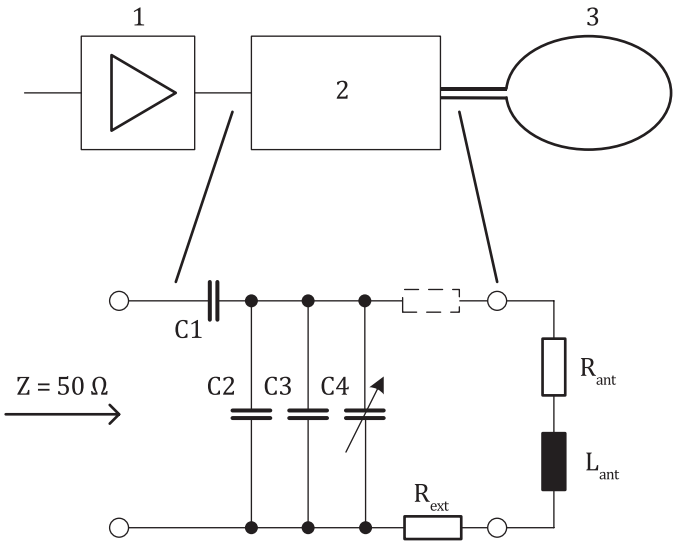
NOTE 1 The tolerance of the matched antenna impedance is ±5 Ω and ±10°.

NOTE 2 R_{ext} is placed on the ground side of the antenna coil.

NOTE 3 The power and voltage ranges include a safety margin.

The linear low distortion variable output 50 Ω power driver should be capable of emitting Type A and Type B modulations for transmission of REQA and REQB as defined in 7.2.3. The output power should be adjustable to deliver *H* fields in the range of 1,5 A/m (rms) to 12 A/m (rms). Care should be taken with the duration of fields above the upper operating range of 7,5 A/m (rms).

A.2.2 Impedance matching network for a bit rate of $f_c/128$



Components list

	Value	Unit	Remarks
C1	47	pF	Voltage range 200 V
C2	180	pF	Voltage range 200 V
C3	22	pF	Voltage range 200 V
C4	2-27	pF	Voltage range 200 V
R_{ext}	0,94	Ω	Power range 10 W

Key

1 50 Ω power driver

2 impedance matching network

3 antenna coil

Figure A.5 — Impedance matching network for a bit rate of $f_c/128$

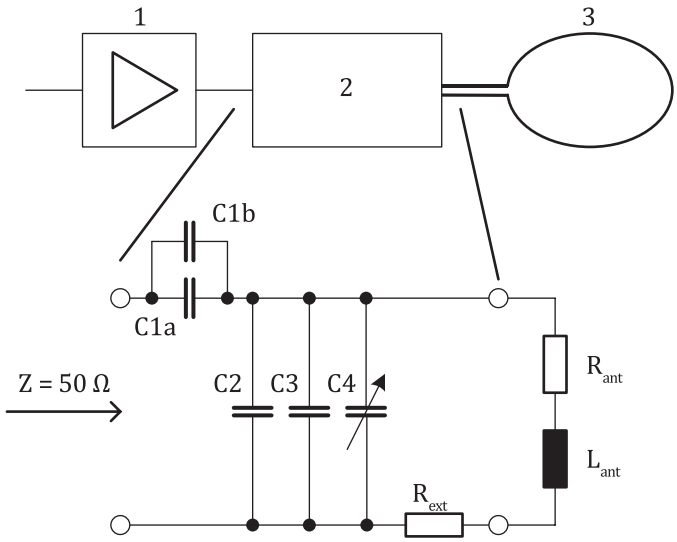
NOTE 1 R_{ext} can be built by connecting five resistors of 4,7 Ω 2 W in parallel.

NOTE 2 R_{ext} can still be placed at the position marked with a dashed outline, as shown in Figure A.5.

NOTE 3 R_{ext} can be 4 W if the maximum field is up to 7,5 A/m (rms).

NOTE 4 The parasitic capacitance of the antenna is not shown Figure A.5.

A.2.3 Impedance matching network for bit rates higher than $f_c/128$



Components list

	Value	Unit	Remarks
C1a	82	pF	Voltage range 200 V
C1b	8,2	pF	Voltage range 200 V
C2	150	pF	Voltage range 200 V
C3	10	pF	Voltage range 200 V
C4	2-27	pF	Voltage range 200 V
R_{ext}	4,7	Ω	Power range 20 W

Key

1 50 Ω power driver
2 impedance matching network
3 antenna coil

Figure A.6 — Impedance matching network for bit rates higher than $f_c/128$

NOTE 1 R_{ext} can be built by a parallel circuit of each two resistors of 4,7 Ω 5 W in series.

R_{ext} should be placed on the ground side of the antenna as drawn in Figure A.6.

NOTE 2 The parasitic capacitance of the antenna is not shown in Figure A.6.

A.3 Test PCD antenna 2

A.3.1 Test PCD antenna 2 layout including impedance matching network

Figure A.7 and Figure A.8 illustrate the Test PCD antenna 2 layout.

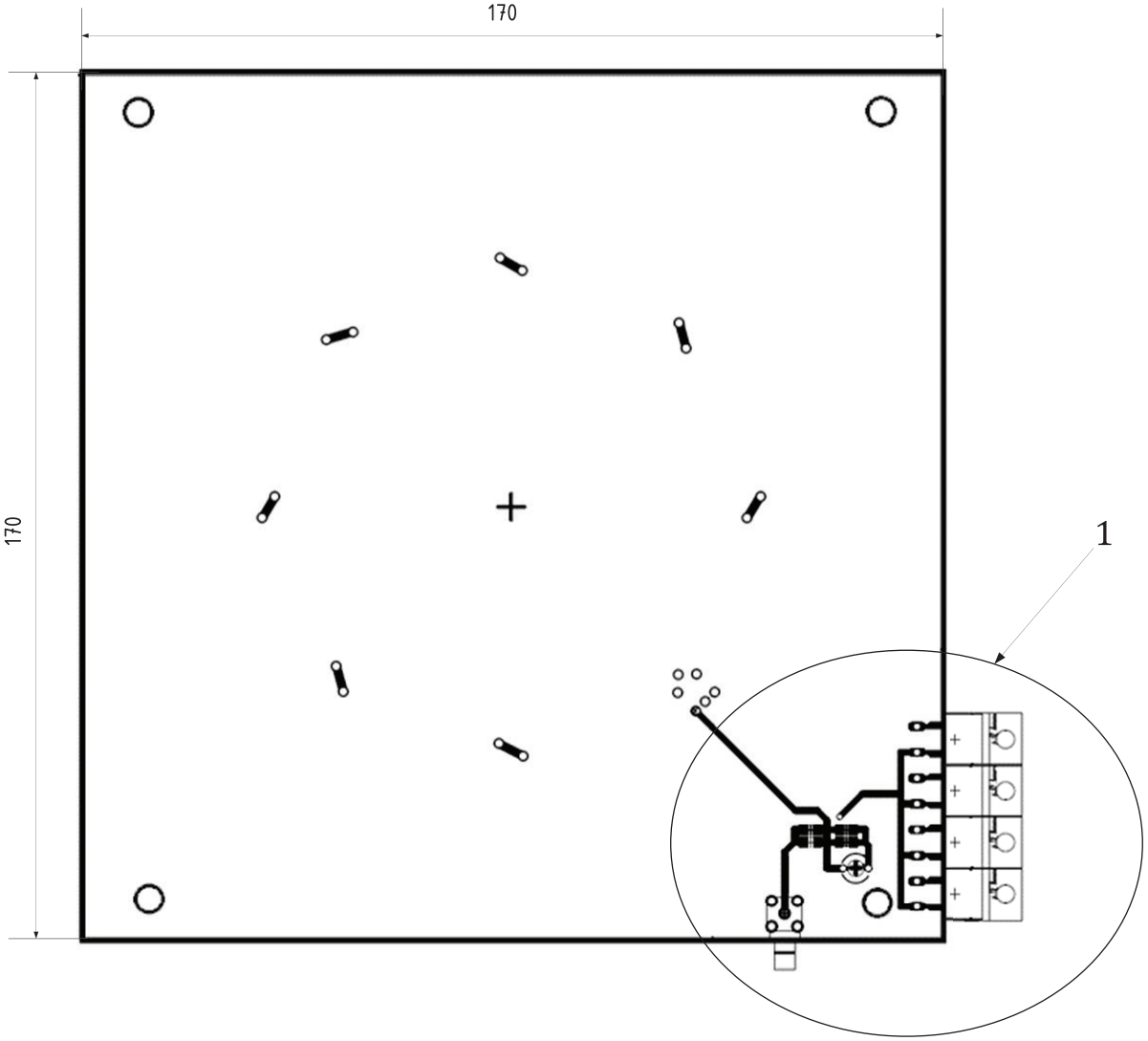
The antenna coil track width is 1,8 mm (except for through-plated holes).

Starting from the impedance matching network there are crossovers every 45°.

Printed circuit board (PCB): FR4 material, thickness 1,6 mm, double sided with 35 µm copper.

NOTE The layout and the position of the impedance matching network are informative.

Dimensions in millimetres



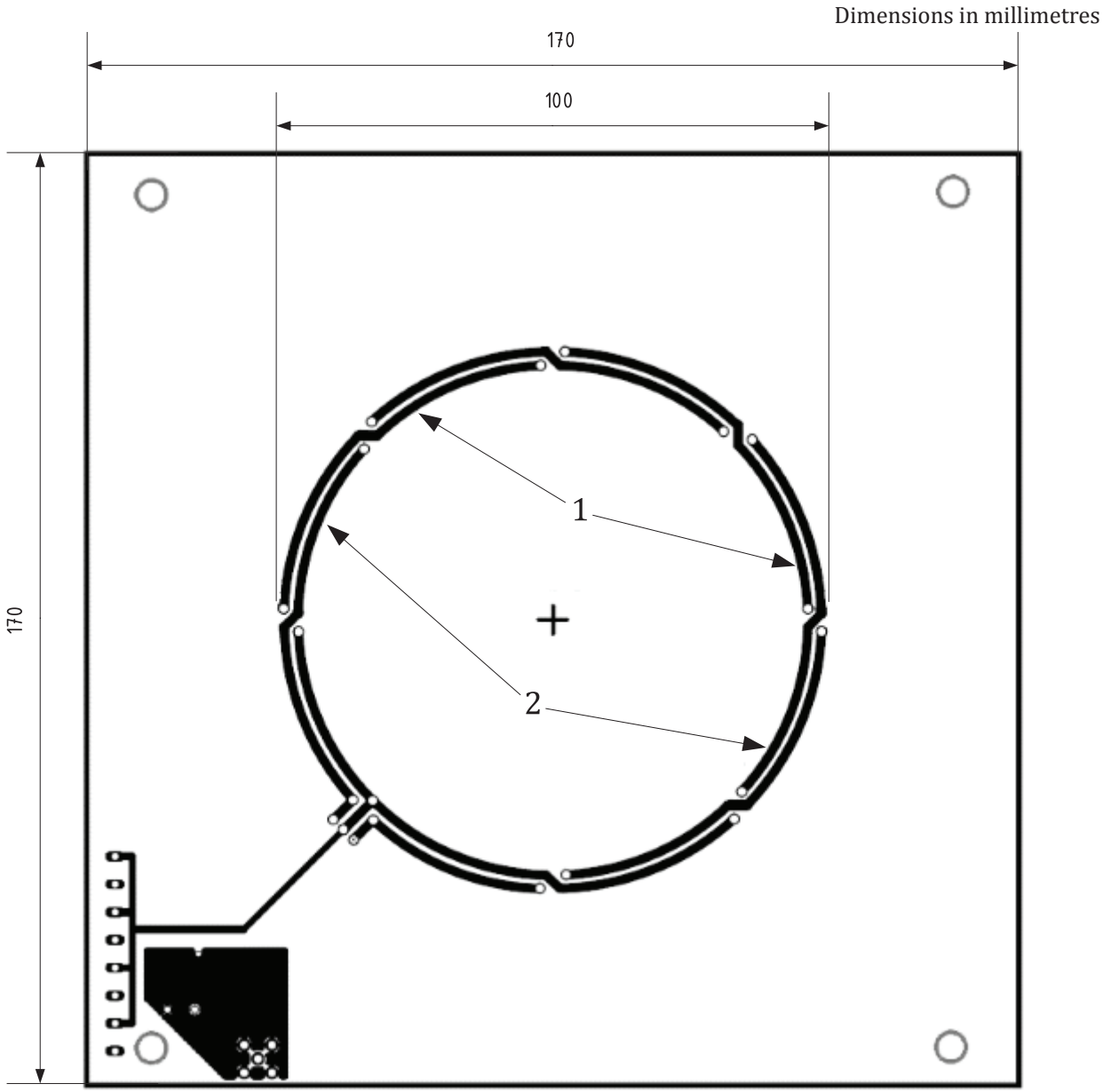
Key

1 impedance matching network

NOTE 1 PCB outside dimensions are informative.

NOTE 2 Drawing is not to scale.

Figure A.7 — Test PCD antenna 2 layout including impedance matching network (view from front)



Key

1 antenna coil

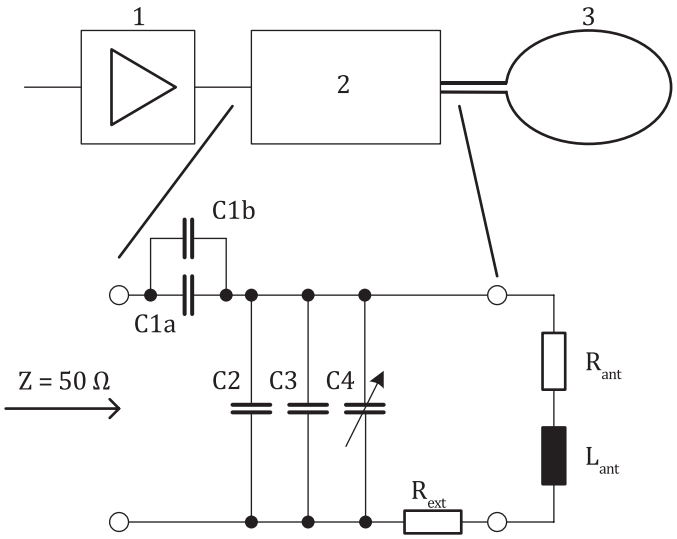
2 ground compensation coil

NOTE 1 PCB outside dimensions are informative.

NOTE 2 Drawing is not to scale.

Figure A.8 — Test PCD antenna 2 layout including impedance matching network (view from back)

A.3.2 Impedance matching network 2



Components list

	Value	Unit	Remarks
C1a	100	pF	Voltage range 200 V
C1b	12	pF	Voltage range 200 V
C2	270	pF	Voltage range 200 V
C3	18	pF	Voltage range 200 V
C4	2-27	pF	Voltage range 200 V
R_{ext}	2,7	Ω	Power range 20 W

Key

1 50 Ω power driver

2 impedance matching network

3 antenna coil

Figure A.9 — Impedance matching network 2

NOTE 1 R_{ext} can be built by either a parallel circuit composed of two equal branches having two resistors of 2,7 Ω 5 W in series each or a parallel circuit of 10 Ω, 10 Ω, 10 Ω and 15 Ω, 5 W.

R_{ext} should be placed on the ground side of the antenna as drawn in Figure A.9.

NOTE 2 The parasitic capacitance of the antenna is not shown in Figure A.9.

Annex B
(informative)

Test PCD Antenna tuning

Figure B.1 and Figure B.2 show the two steps of a simple phase tuning procedure to match the impedance of the antenna to that of the driving generator. After the two steps of the tuning procedure the signal generator shall be directly connected to the antenna output for the tests.

Step 1:

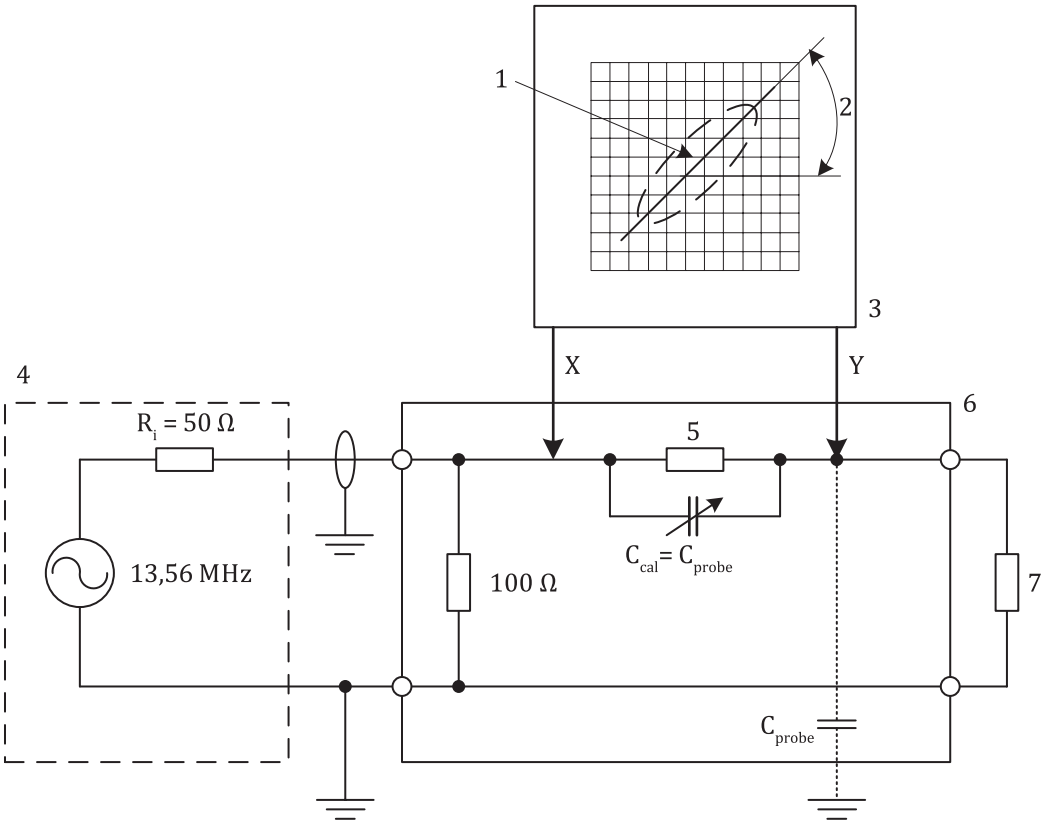
A high precision resistor of 50 Ω with a tolerance of ±1 % (e.g. 50 Ω BNC resistor) is inserted in the signal line between the signal generator output and an antenna connector. The two probes of the oscilloscope are connected to both sides of the serial reference resistor. The oscilloscope displays a Lissajous figure when it is set in Y to X presentation. The signal generator is set to:

1) wave form: sinusoidal;

2) frequency: 13,56 MHz;

3) amplitude: 2 V (rms) to 5 V (rms).

The output is terminated with a second high precision resistor of 50 Ω with a tolerance of ±1 % (e.g. 50 Ω BNC terminating resistor). The probe, which is in parallel to the output connector has a small parasitic capacitance C_{probe}. A calibration capacitance C_{cal} in parallel to the reference resistor compensates this probe capacitor if $C_{cal} = C_{probe}$. The probe capacitor is compensated when the Lissajous figure is completely closed.



Key

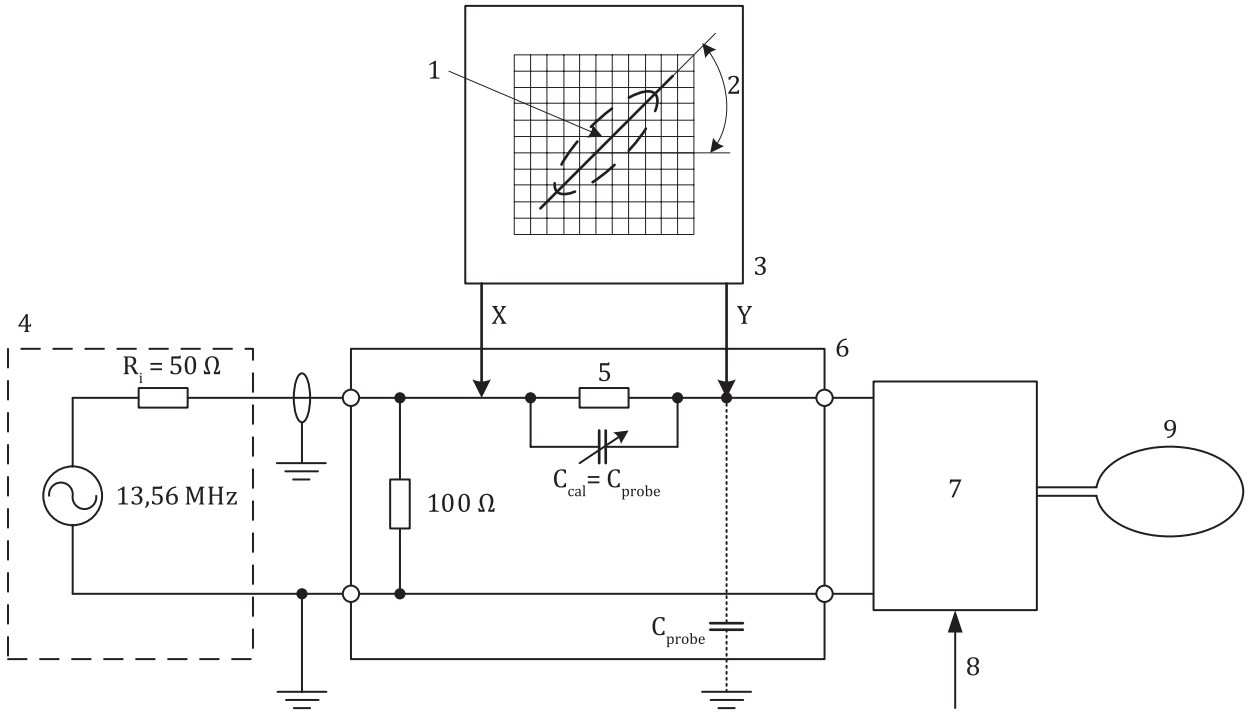
1 closed figure $\varphi = 0$
2 angle corresponding to 50 Ω
3 oscilloscope
4 signal generator
5 reference: 50 Ω
6 output
7 calibration resistor 50 Ω

Figure B.1 — Calibration set-up (Step 1)

NOTE The ground cable needs to be run close to the probe to avoid induced voltages caused by the magnetic field.

Step 2:

Using the same values as set for step 1, in the second step the matching circuitry is connected to the antenna output. The capacitor C4 on the antenna board is used to tune the phase to zero.



Key

1 closed figure $\varphi = 0$
2 angle corresponding to 50 Ω
3 oscilloscope
4 signal generator
5 reference: 50 Ω
6 output
7 impedance matching network
8 C4 phase calibration
9 antenna coil

Figure B.2 — Calibration set-up (Step 2)

Annex C
(normative)

Sense coil

C.1 Sense coil layout

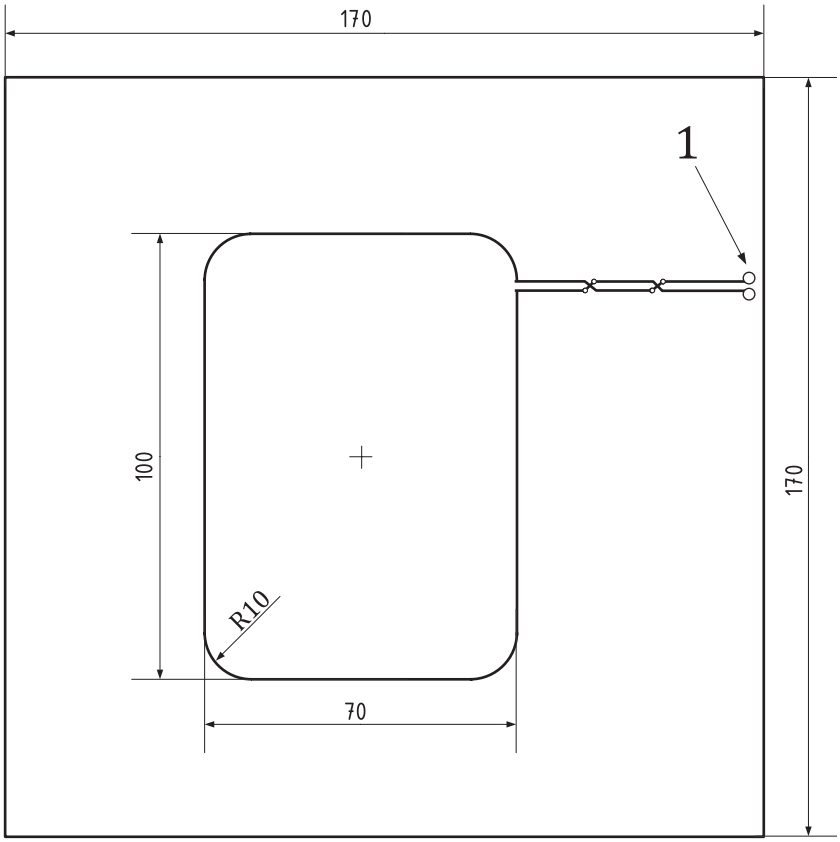
C.1.1 Sense coil 1 layout

Figure C.1 illustrates sense coils 1 layout.

The sense coil track width is 0,5 mm with relative tolerance ±20 % (except for through-plated holes). Size of the coils refers to the outer dimensions.

Printed circuit board (PCB): FR4 material, thickness 1,6 mm, double sided with 35 µm copper.

Dimensions in millimetres



Key

1 connections

NOTE 1 PCB outside dimensions are informative.

NOTE 2 Drawing is not to scale.

Figure C.1 — Layout for sense coils 1 (a and b)

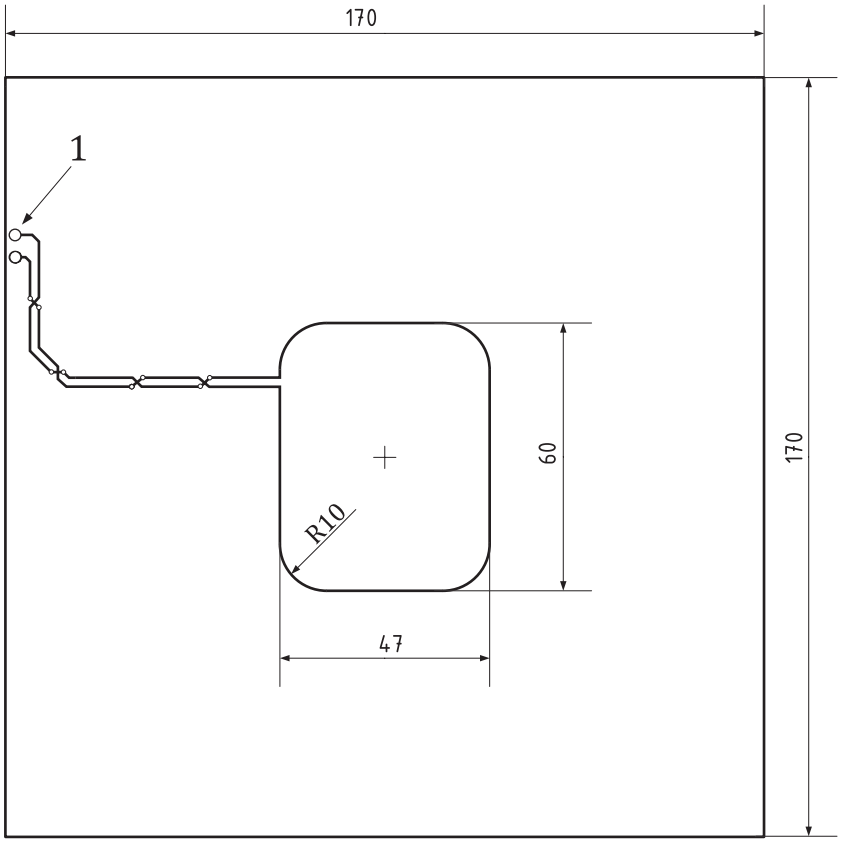
C.1.2 Sense coil 2 layout

Figure C.2 illustrates the sense coils 2 layout.

The sense coil track width is 0,5 mm with relative tolerance ±20 % (except for through-plated holes). Size of the coils refers to the outer dimensions.

Printed circuit board (PCB): FR4 material, thickness 1,6 mm, double sided with 35 µm copper.

Dimensions in millimetres



Key

1 connections

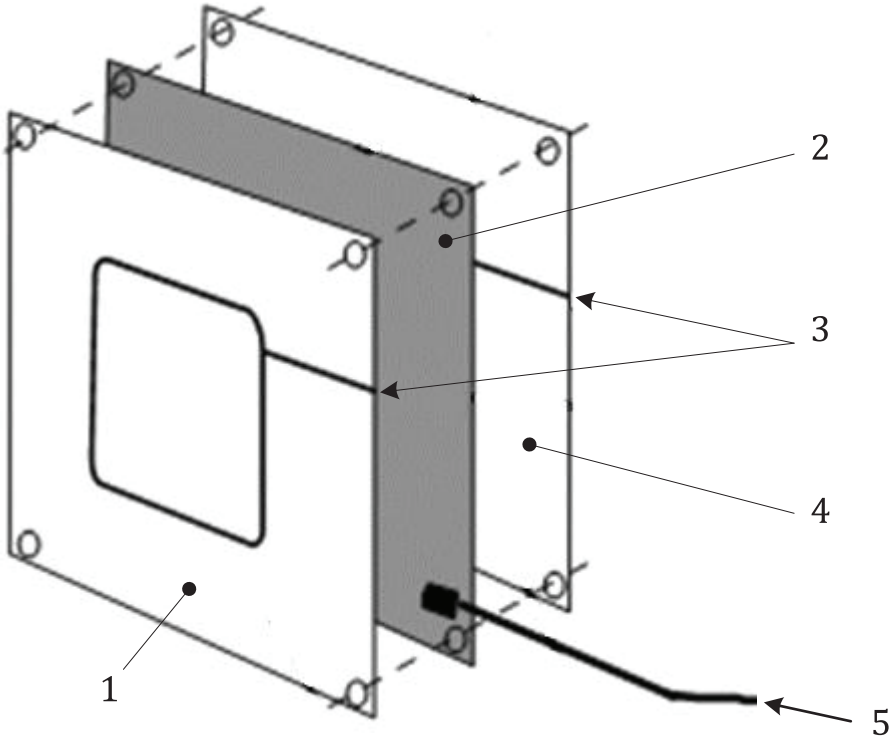
NOTE 1 PCB outside dimensions are informative.

NOTE 2 Drawing is not to scale.

Figure C.2 — Layout for sense coils 2 (a and b)

C.1.3 Sense coil assembly

Figure C.3 illustrates the sense coil assembly.



Key

1 sense coil b
2 test PCD antenna
3 connections
4 sense coil a
5 13,56 MHz signal

Figure C.3 — Sense coil assembly

Annex D
(normative)

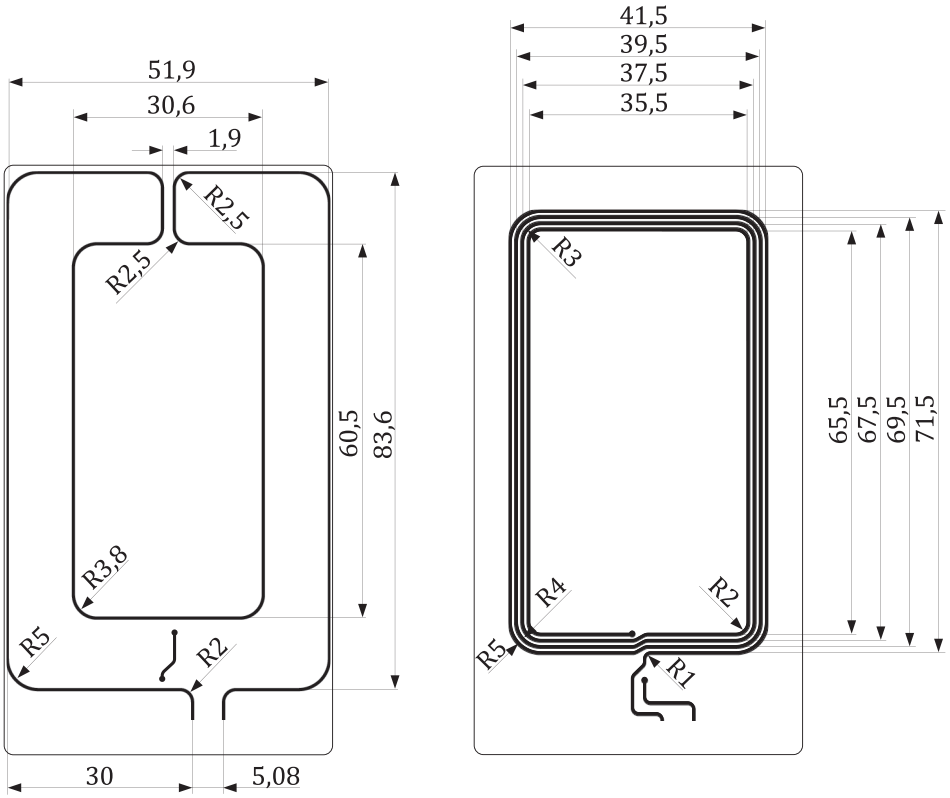
Reference PICCs and Active Reference PICCs

D.1 Reference PICCs

D.1.1 Reference PICC 1 coil layouts

Figure D.1 specifies the Reference PICC 1 pick up coil and main coil layouts.

Dimensions in millimetres



View from front, pick up coil **View from back, main coil**

NOTE 1 Dimensions of coil tracks refer to coil track center.

NOTE 2 Drawing is not to scale.

Figure D.1 — Reference PICC 1 pick up coil and main coil layouts

The pick up coil and the main coil shall be concentric.

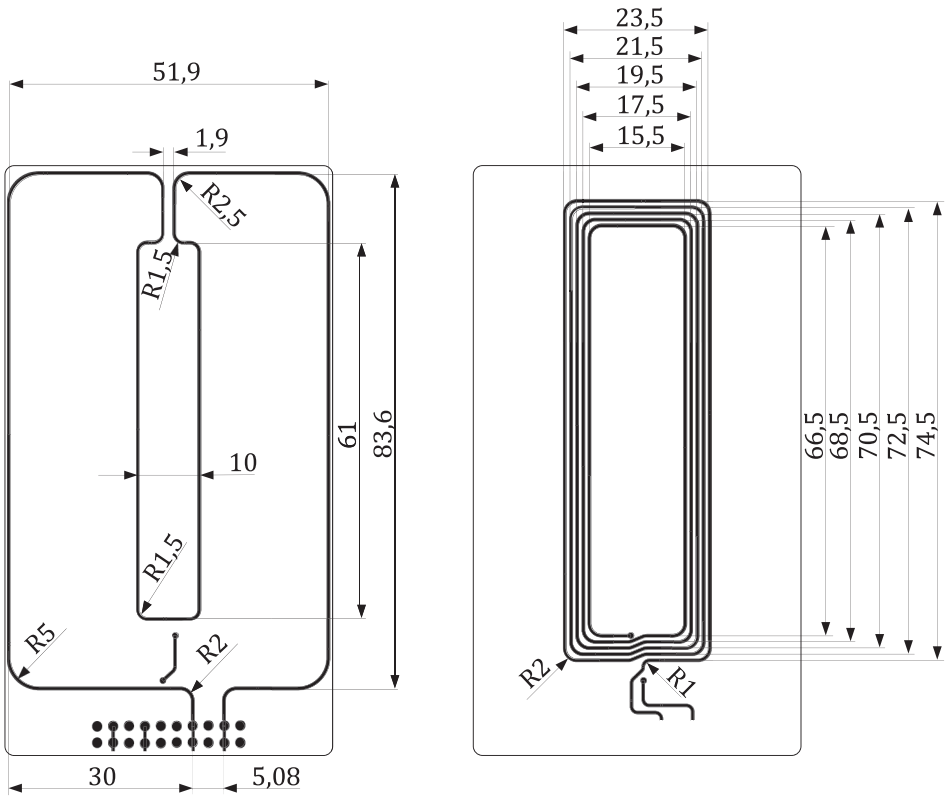
The two coils track width and spacing shall be 0,5 mm with a relative tolerance of ±20 %.

Printed circuit board (PCB): FR4 material, thickness 0,76 mm with a relative tolerance of ±10 %, double sided with 35 µm copper.

D.1.2 Reference PICC 2 coil layouts

Figure D.2 specifies the Reference PICC 2 pick up coil and main coil layouts.

Dimensions in millimetres



View from front, pick up coil **View from back, main coil**

NOTE 1 Dimensions of coil tracks refer to coil track center.

NOTE 2 Drawing is not to scale.

Figure D.2 — Reference PICC 2 pick up coil and main coil layouts

The pick up coil and the main coil shall be concentric.

The two coils track width and spacing shall be 0,5 mm with a relative tolerance of ±20 %.

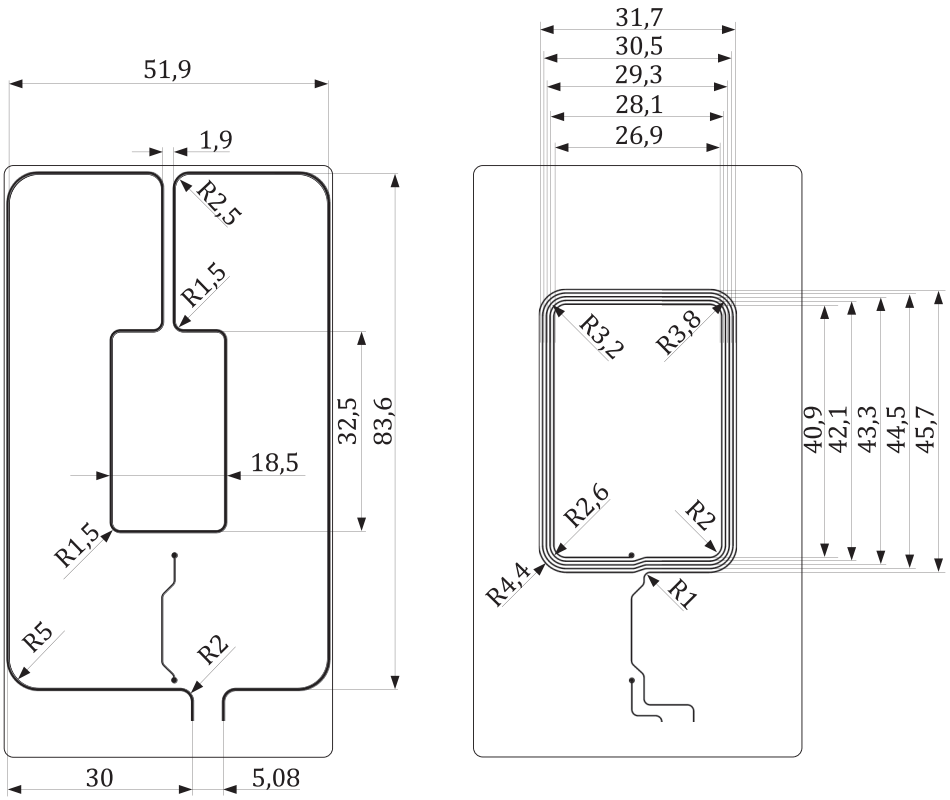
All main coil corners radii shall be 2 mm.

Printed circuit board (PCB): FR4 material, thickness 0,76 mm with a relative tolerance of ±10 %, double sided with 35 µm copper.

D.1.3 Reference PICC 3 coil layouts

Figure D.3 specifies the Reference PICC 3 pick up coil and main coil layouts.

Dimensions in millimetres



View from front, pick up coil **View from back, main coil**

NOTE 1 Dimensions of coil tracks refer to coil track center.

NOTE 2 Drawing is not to scale.

Figure D.3 — Reference PICC 3 pick up coil and main coil layouts

The pick up coil and the main coil shall be concentric.

The pick up coil track width shall be 0,5 mm with a relative tolerance of ±20 %.

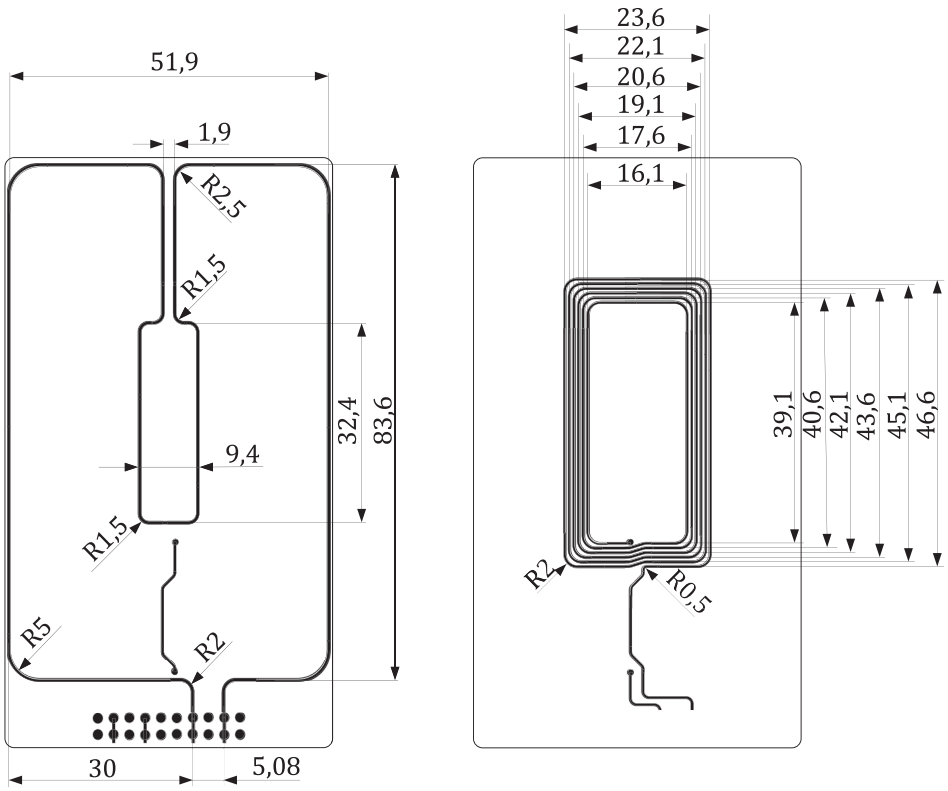
The main coil track width and spacing shall be 0,3 mm with a relative tolerance of ±20 %.

Printed circuit board (PCB): FR4 material, thickness 0,76 mm with a relative tolerance of ±10 %, double sided with 35 µm copper.

D.1.4 Reference PICC 4 coil layouts

Figure D.4 specifies the Reference PICC 4 pick up coil and main coil layouts.

Dimensions in millimetres



View from front, pick up coil **View from back, main coil**

NOTE 1 Dimensions of coil tracks refer to coil track center.

NOTE 2 Drawing is not to scale.

Figure D.4 — Reference PICC 4 pick up coil and main coil layouts

The pick up coil and the main coil shall be concentric.

The pick up coil track width shall be 0,5 mm with a relative tolerance of ±20 %.

All main coil corners radii shall be 2 mm.

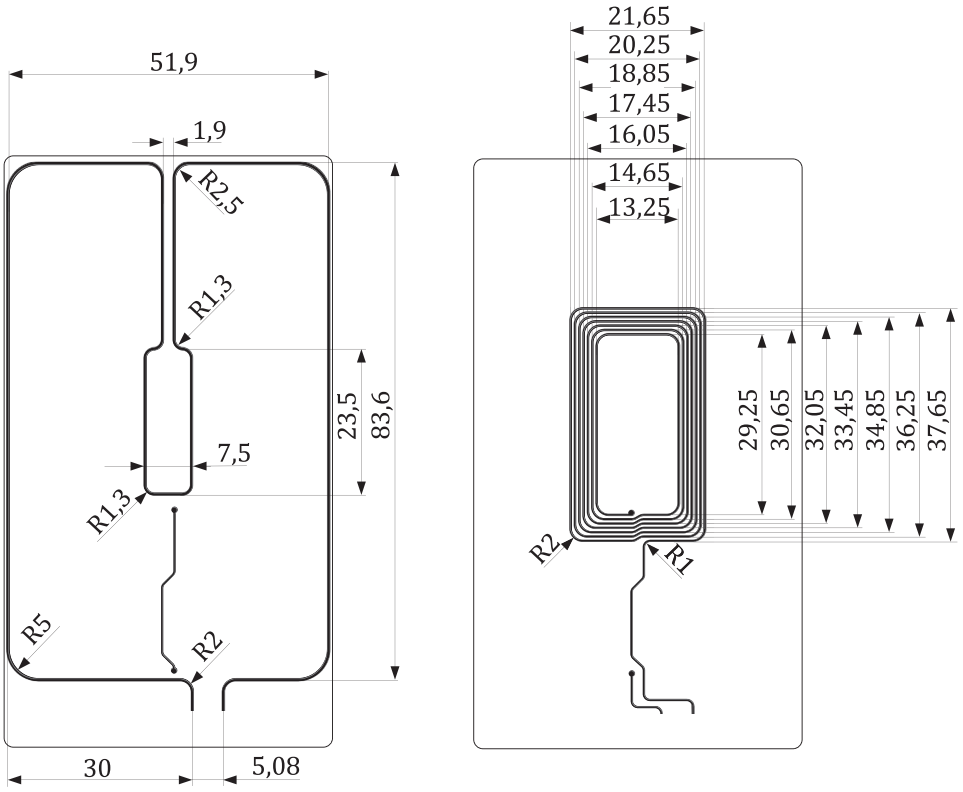
The main coil track width shall be 0,4 mm and the spacing shall be 0,35 mm with a relative tolerance of ±20 %.

Printed circuit board (PCB): FR4 material, thickness 0,76 mm with a relative tolerance of ±10 %, double sided with 35 µm copper.

D.1.5 Reference PICC 5 coil layouts

Figure D.5 specifies the Reference PICC 5 pick up coil and main coil layouts.

Dimensions in millimetres



View from front, pick up coil **View from back, main coil**

NOTE 1 Dimensions of coil tracks refer to coil track center.

NOTE 2 Drawing is not to scale.

Figure D.5 — Reference PICC 5 pick up coil and main coil layouts

The pick up coil and the main coil shall be concentric.

The pick up coil track width shall be 0,5 mm with a relative tolerance of ±20 %.

All main coil corners radii shall be 2 mm.

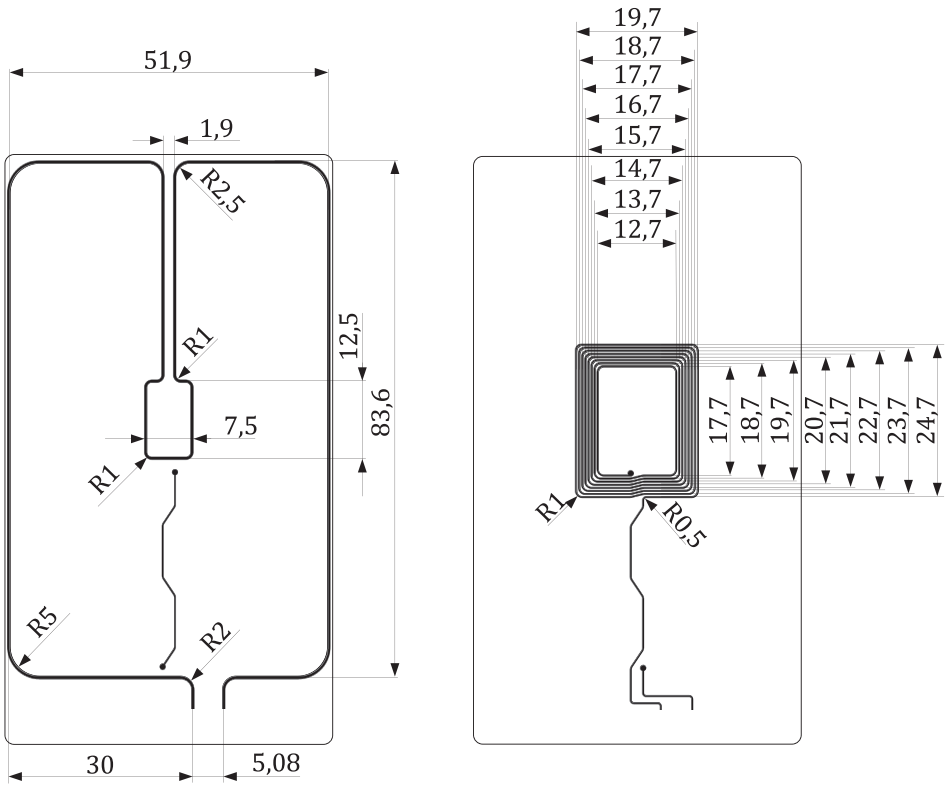
The main coil track width and spacing shall be 0,35 mm with a relative tolerance of ±20 %.

Printed circuit board (PCB): FR4 material, thickness 0,76 mm with a relative tolerance of ±10 %, double sided with 35 µm copper.

D.1.6 Reference PICC 6 coil layouts

Figure D.6 specifies the Reference PICC 6 pick up coil and main coil layouts.

Dimensions in millimetres



View from front, pick up coil **View from back, main coil**

NOTE 1 Dimensions of coil tracks refer to coil track center.

NOTE 2 Drawing is not to scale.

Figure D.6 — Reference PICC 6 pick up coil and main coil layouts

The pick up coil and the main coil shall be concentric.

The pick up coil track width shall be 0,5 mm with a relative tolerance of ±20 %.

All main coil corners radii shall be 1 mm.

The main coil track width shall be 0,3 mm and the spacing shall be 0,2 mm with a relative tolerance of ±20 %.

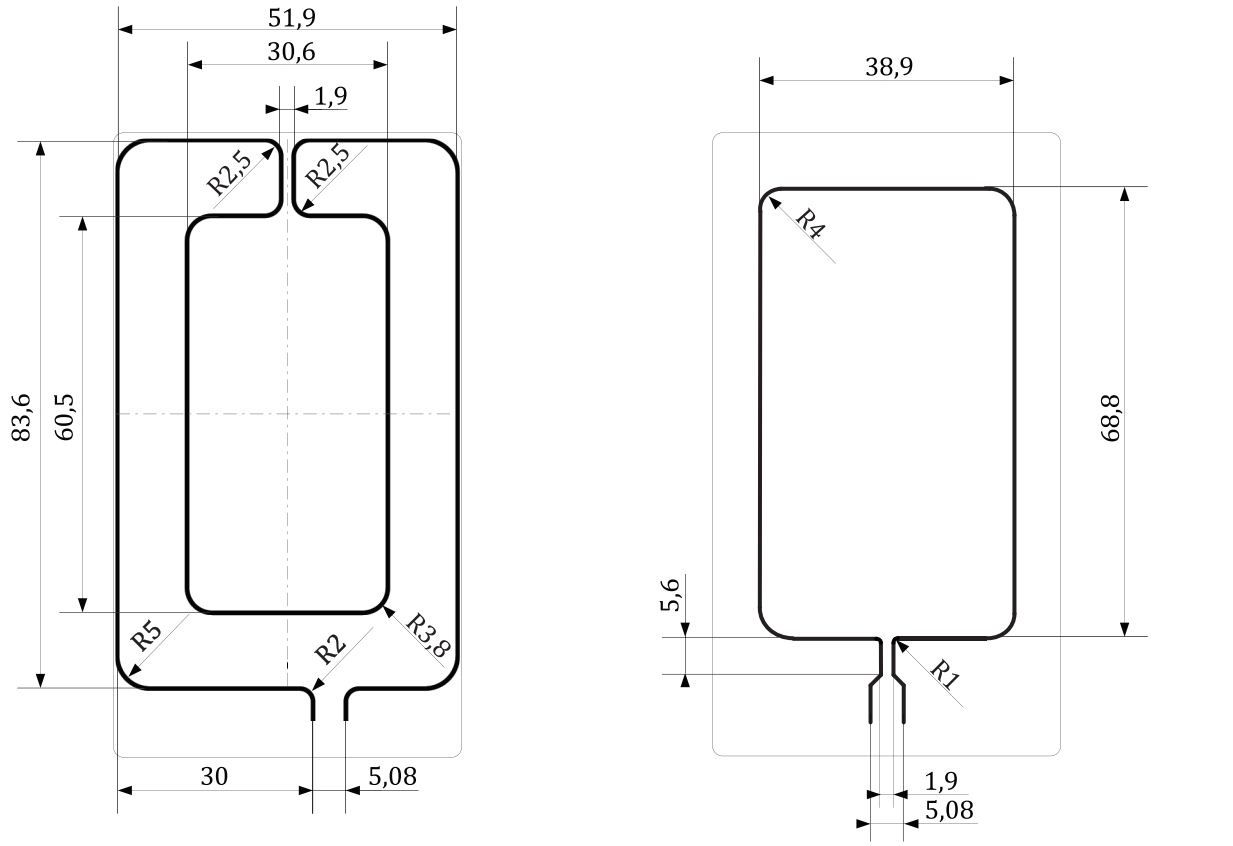
Printed circuit board (PCB): FR4 material, thickness 0,76 mm with a relative tolerance of ±10 %, double sided with 35 µm copper.

D.2 Active Reference PICCs

D.2.1 Active Reference PICC 1 coil layouts

Figure D.7 specifies the Active Reference PICC 1 pick up coil and main coil layouts.

Dimensions in millimetres



View from front, pick up coil **View from back, main coil**

NOTE 1 Dimensions of coil tracks refer to coil track center.

NOTE 2 Drawing is not to scale.

Figure D.7 — Active Reference PICC 1 pick up coil and main coil layouts

The pick up coil and the main coil shall be concentric.

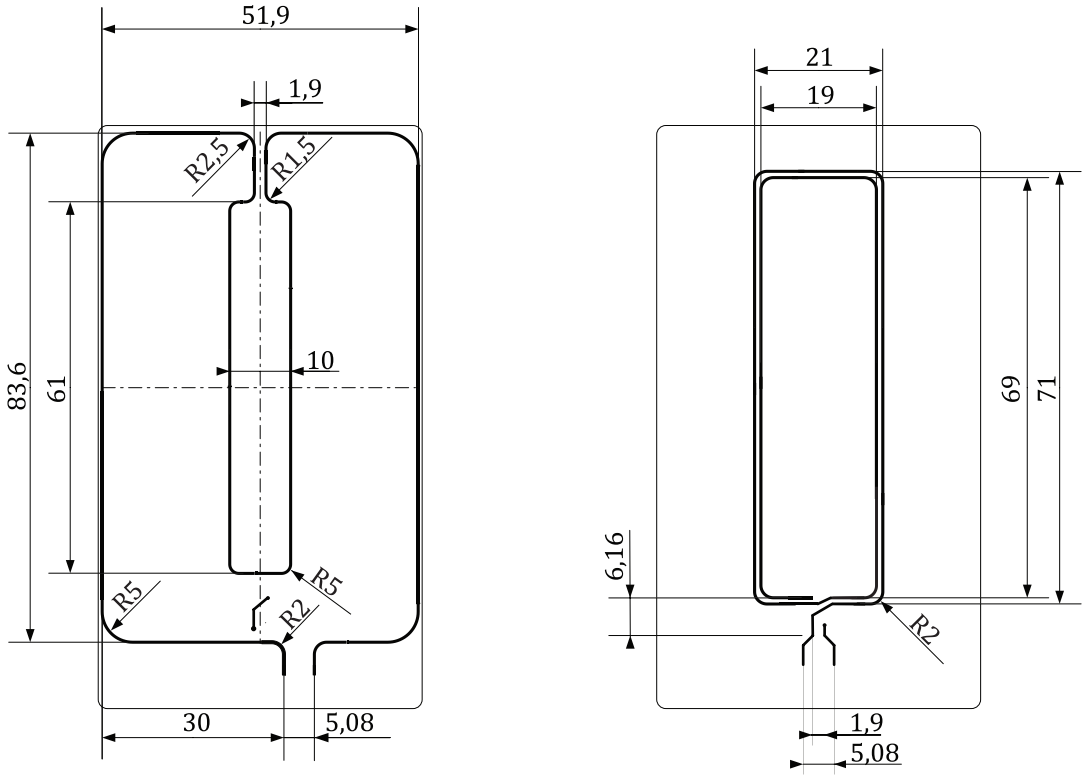
The two coils track width and the spacing shall be 0,5 mm with a relative tolerance of ±20 %.

Printed circuit board (PCB): FR4 material, thickness 0,76 mm with a relative tolerance of ±10 %, double sided with 35 µm copper.

D.2.2 Active Reference PICC 2 coil layouts

Figure D.8 specifies the Active Reference PICC 2 pick up coil and main coil layouts.

Dimensions in millimetres



View from front, pick up coil **View from back, main coil**

NOTE 1 Dimensions of coil tracks refer to coil track center.

NOTE 2 Drawing is not to scale.

Figure D.8 — Active Reference PICC 2 pick up coil and main coil layouts

The pick up coil and the main coil shall be concentric.

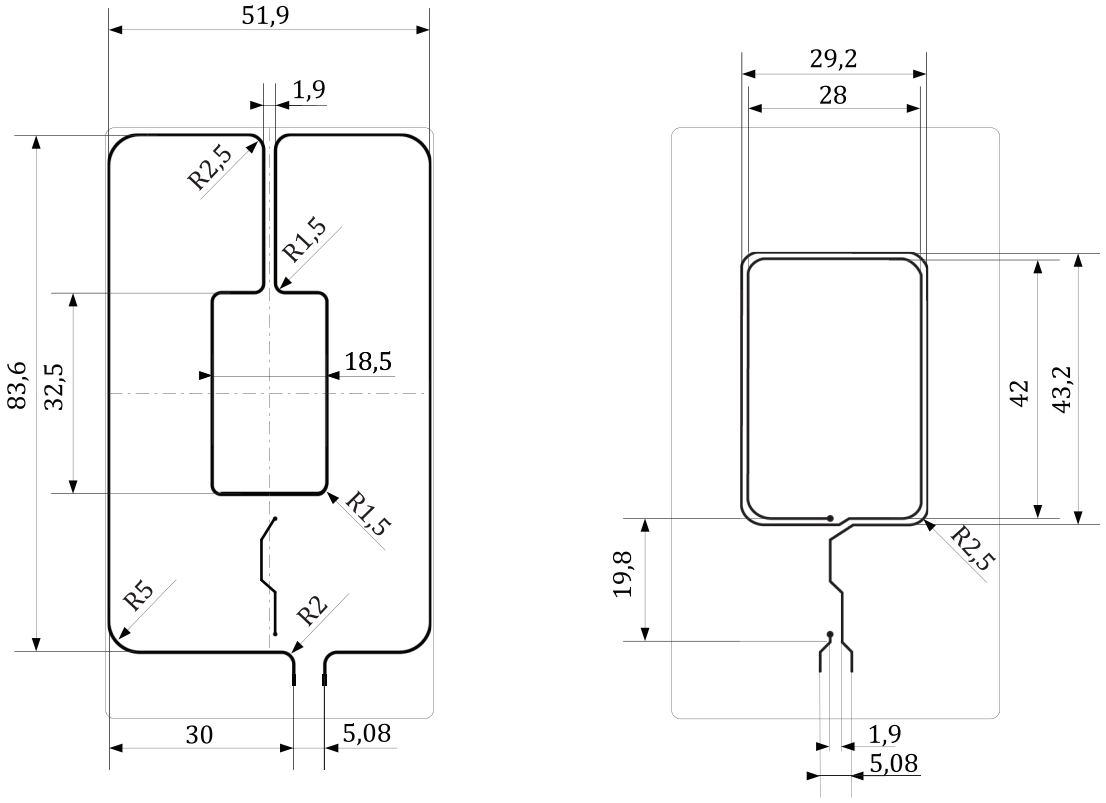
The two coils track width and the spacing shall be 0,5 mm with a relative tolerance of ±20 %.

Printed circuit board (PCB): FR4 material, thickness 0,76 mm with a relative tolerance of ±10 %, double sided with 35 µm copper.

D.2.3 Active Reference PICC 3 coil layouts

Figure D.9 specifies the Active Reference PICC 3 pick up coil and main coil layouts.

Dimensions in millimetres



View from front, pick up coil **View from back, main coil**

NOTE 1 Dimensions of coil tracks refer to coil track center.

NOTE 2 Drawing is not to scale.

Figure D.9 — Active Reference PICC 3 pick up coil and main coil layouts

The pick up coil and the main coil shall be concentric.

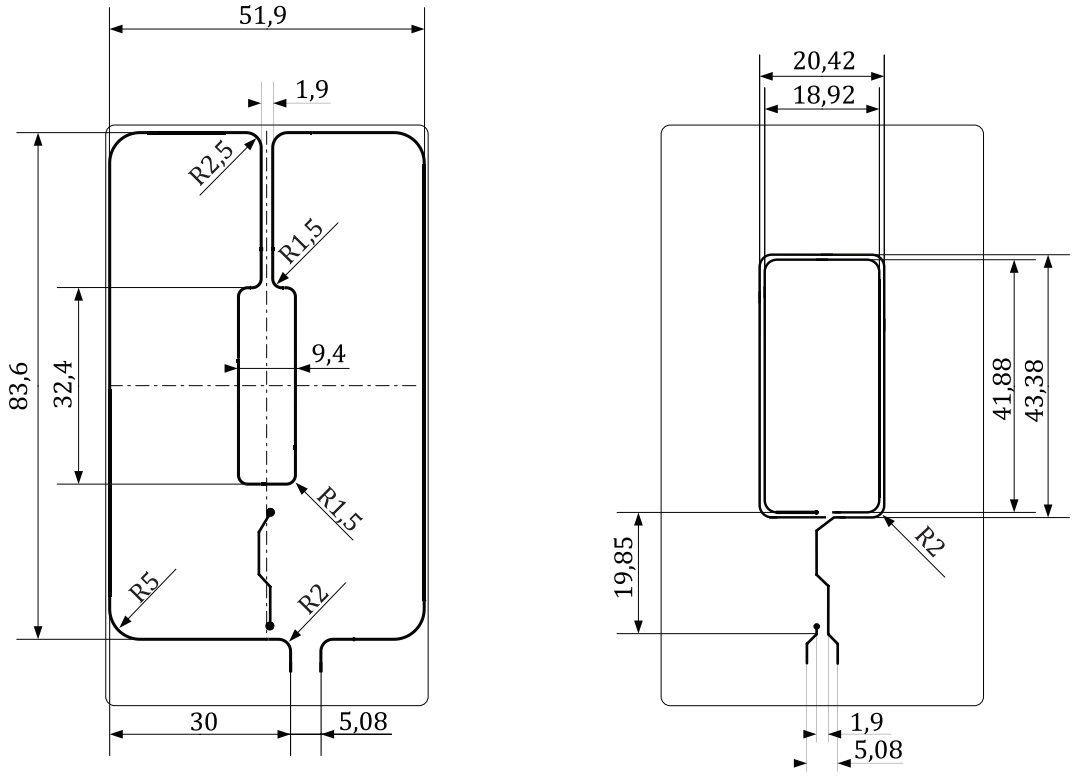
The two coils track width and the spacing shall be 0,3 mm with a relative tolerance of ±20 %.

Printed circuit board (PCB): FR4 material, thickness 0,76 mm with a relative tolerance of ±10 %, double sided with 35 µm copper.

D.2.4 Active Reference PICC 4 coil layouts

Figure D.10 specifies the Active Reference PICC 4 pick up coil and main coil layouts.

Dimensions in millimetres



View from front, pick up coil **View from back, main coil**

NOTE 1 Dimensions of coil tracks refer to coil track center.

NOTE 2 Drawing is not to scale.

Figure D.10 — Active Reference PICC 4 pick up coil and main coil layouts

The pick up coil and the main coil shall be concentric.

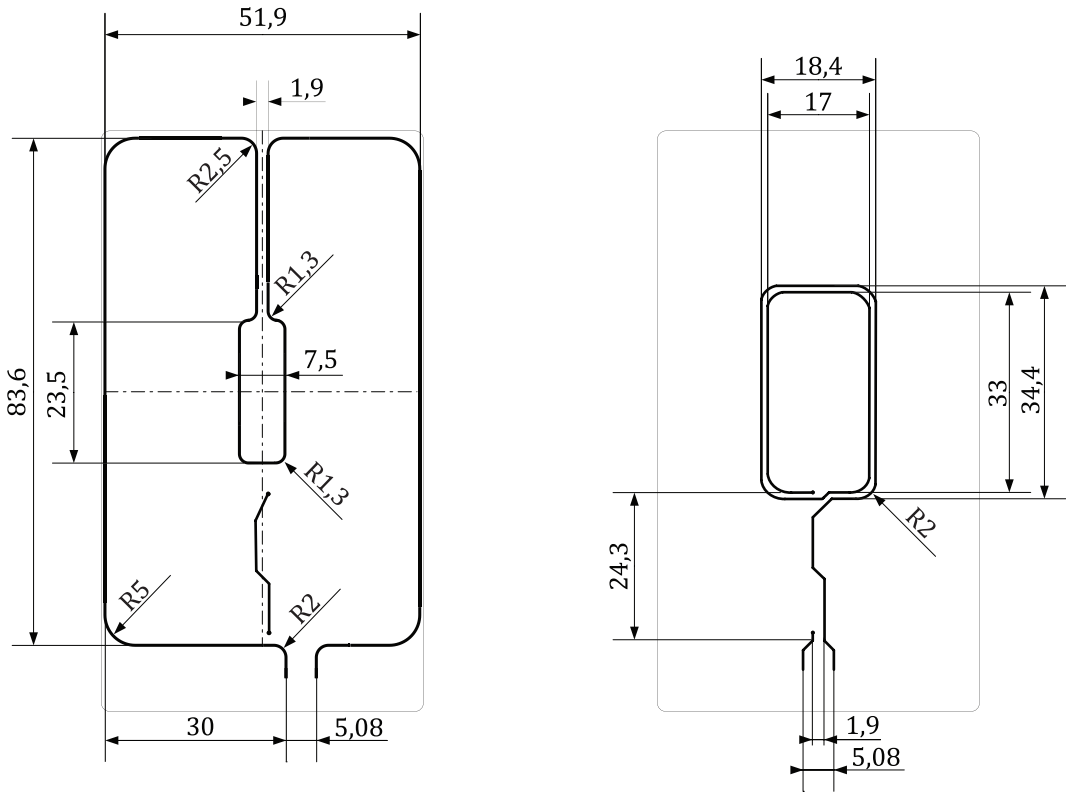
The two coils track width shall be 0,4 mm and the spacing shall be 0,35 mm with a relative tolerance of ±20 %.

Printed circuit board (PCB): FR4 material, thickness 0,76 mm with a relative tolerance of ±10 %, double sided with 35 µm copper.

D.2.5 Active Reference PICC 5 coil layouts

Figure D.11 specifies the Active Reference PICC 5 pick up coil and main coil layouts.

Dimensions in millimetres



View from front, pick up coil **View from back, main coil**

NOTE 1 Dimensions of coil tracks refer to coil track center.

NOTE 2 Drawing is not to scale.

Figure D.11 — Active Reference PICC 5 pick up coil and main coil layouts

The pick up coil and the main coil shall be concentric.

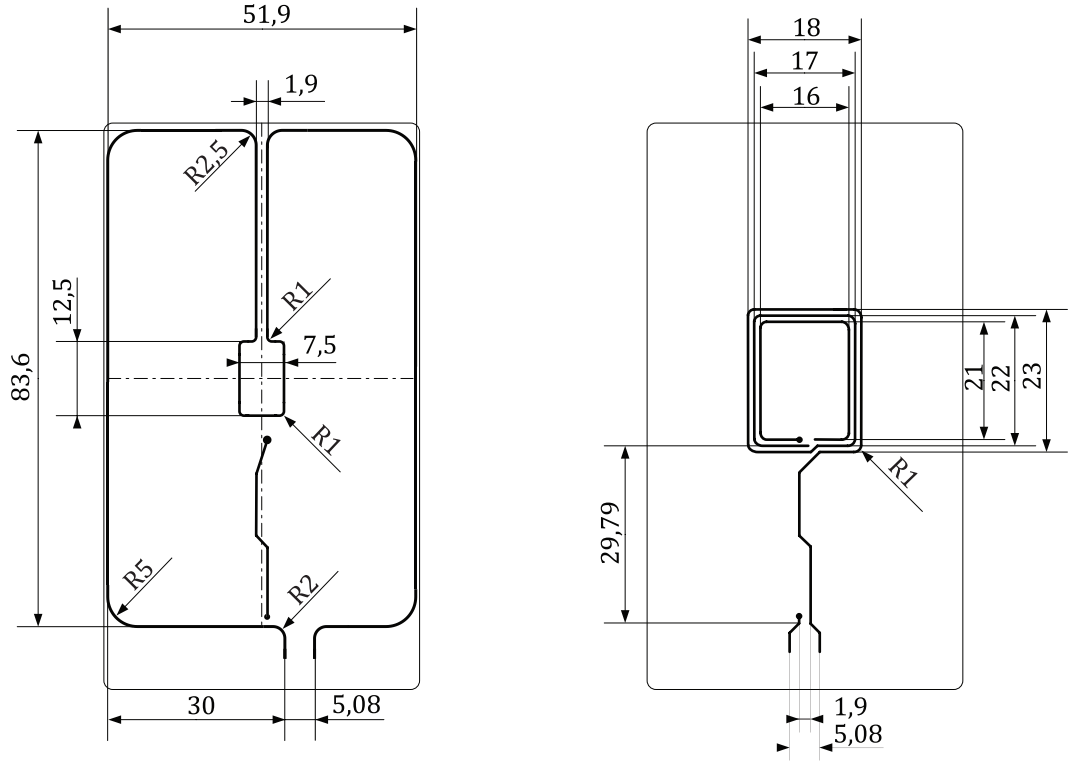
The two coils track width and the spacing shall be 0,35 mm with a relative tolerance of ±20 %.

Printed circuit board (PCB): FR4 material, thickness 0,76 mm with a relative tolerance of ±10 %, double sided with 35 µm copper.

D.2.6 Active Reference PICC 6 coil layouts

Figure D.12 specifies the Active Reference PICC 6 pick up coil and main coil layouts.

Dimensions in millimetres



View from front, pick up coil **View from back, main coil**

NOTE 1 Dimensions of coil tracks refer to coil track center.

NOTE 2 Drawing is not to scale.

Figure D.12 — Active Reference PICC 6 pick up coil and main coil layouts

The pick up coil and the main coil shall be concentric.

The two coils track width shall be 0,3 mm and the spacing shall be 0,2 mm with a relative tolerance of ±20 %.

Printed circuit board (PCB): FR4 material, thickness 0,76 mm with a relative tolerance of ±10 %, double sided with 35 µm copper.

Annex E
(normative)

PCD modulation index *m* and waveform analysis tool

E.1 Overview

The working principle of the PCD modulation index *m* and waveform analysis tool is illustrated in Figure E.1.

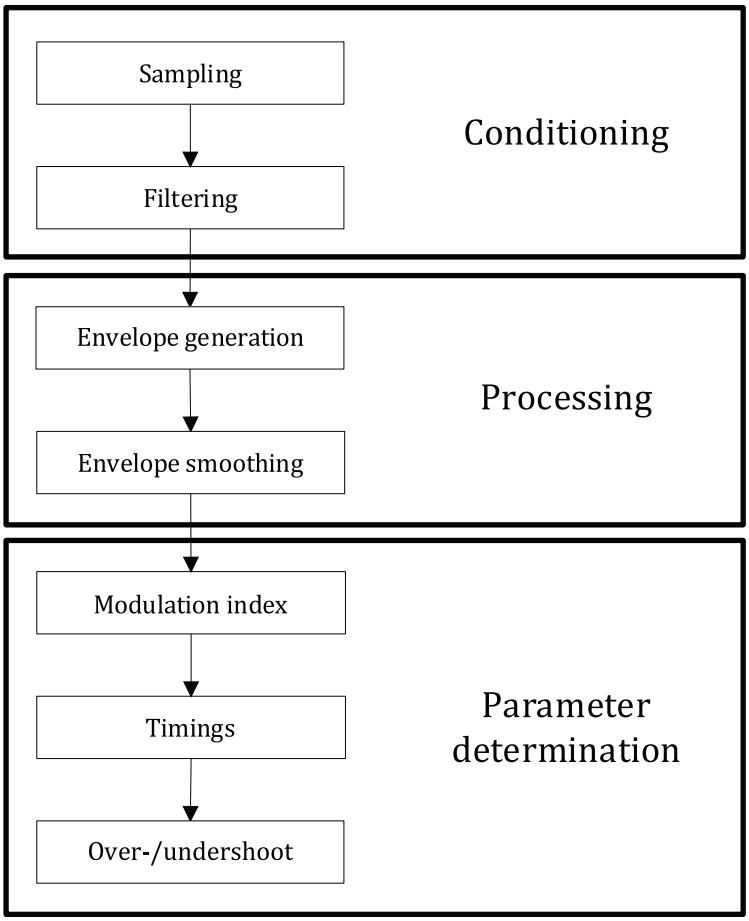


Figure E.1 — PCD modulation index *m* and waveform analysis tool block diagram

Each block is described in the following clauses.

E.2 Sampling

E.2.1 Sampling for PCD to PICC bit rates of $f_c/128$, $f_c/64$, $f_c/32$ and $f_c/16$

The oscilloscope used for signal capturing shall fulfill the requirements defined in 5.2.

The time and voltage data of one modulation pulse (see Figure E.2) shall be transferred to a suitable computer.

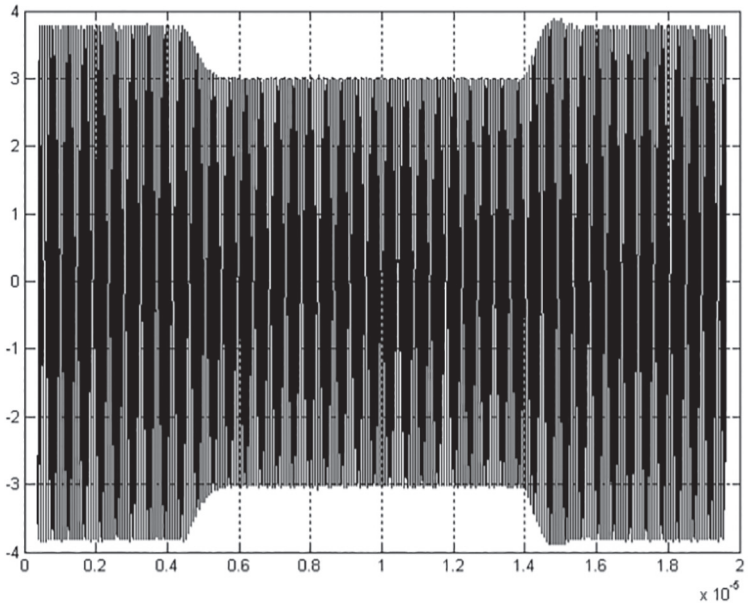


Figure E.2 — Modulation pulse

E.2.2 Sampling for PCD to PICC bit rates of $f_c/8$, $f_c/4$ and $f_c/2$

The oscilloscope used for signal capturing shall fulfill the requirements defined in 5.2.

The time and voltage data of a PCD frame containing short and long modulation pulses (preferably a complete S(DESELECT) command) as illustrated in Figure E.3, with at least 20 carrier periods before the first and after the last modulation pulse, shall be transferred to a suitable computer.

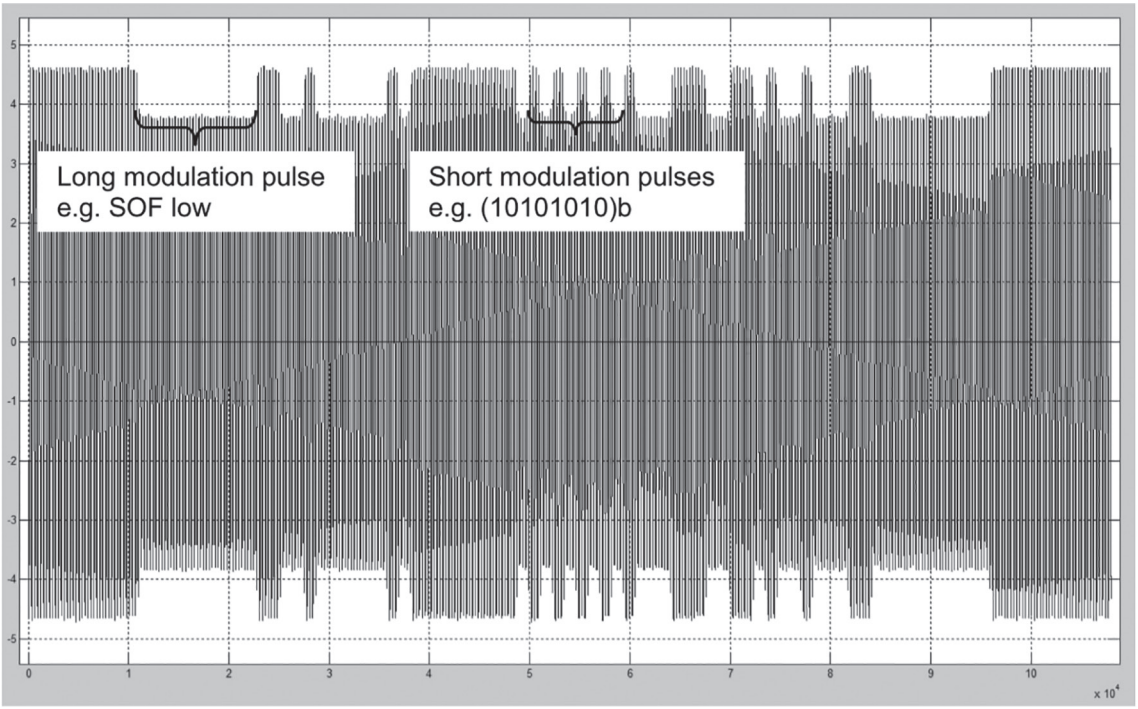


Figure E.3 — Modulation pulses

E.3 Filtering

E.3.1 Filtering for PCD to PICC bit rates of $f_c/128, f_c/64, f_c/32$ and $f_c/16$

A 4th order, Butterworth type band pass filter with center frequency of 13,56 MHz and 10 MHz 3 dB bandwidth shall be used for filtering the DC and higher harmonic components. The filter characteristic is illustrated in Figure E.4.

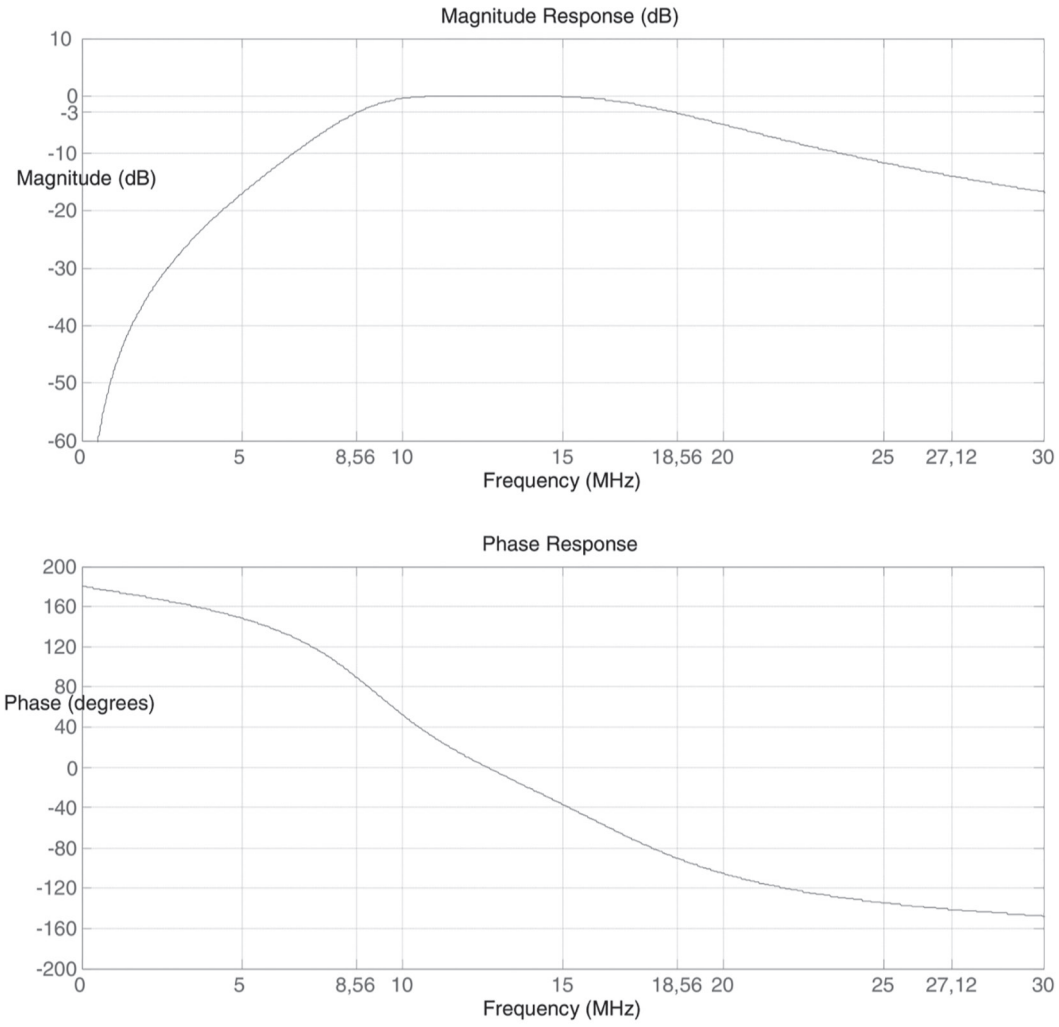


Figure E.4 — Filter characteristics

E.3.2 Filtering for PCD to PICC bit rates of $f_c/8, f_c/4$ and $f_c/2$

A 4th order, Butterworth type band pass filter with center frequency of 13,56 MHz and 15 MHz 3 dB bandwidth shall be used for filtering the DC and higher harmonic components.

E.4 Envelope generation

The filtered signal shall be Hilbert transformed and the magnitude of this complex transform represents the signal envelope.

E.5 Envelope smoothing

E.5.1 Envelope smoothing for PCD to PICC bit rates of $f_c/128, f_c/64, f_c/32$ and $f_c/16$

The signal envelope shall be smoothened with a moving average filter and the filter period shall be one carrier period. The smoothened envelope signal is illustrated in Figure E.5.

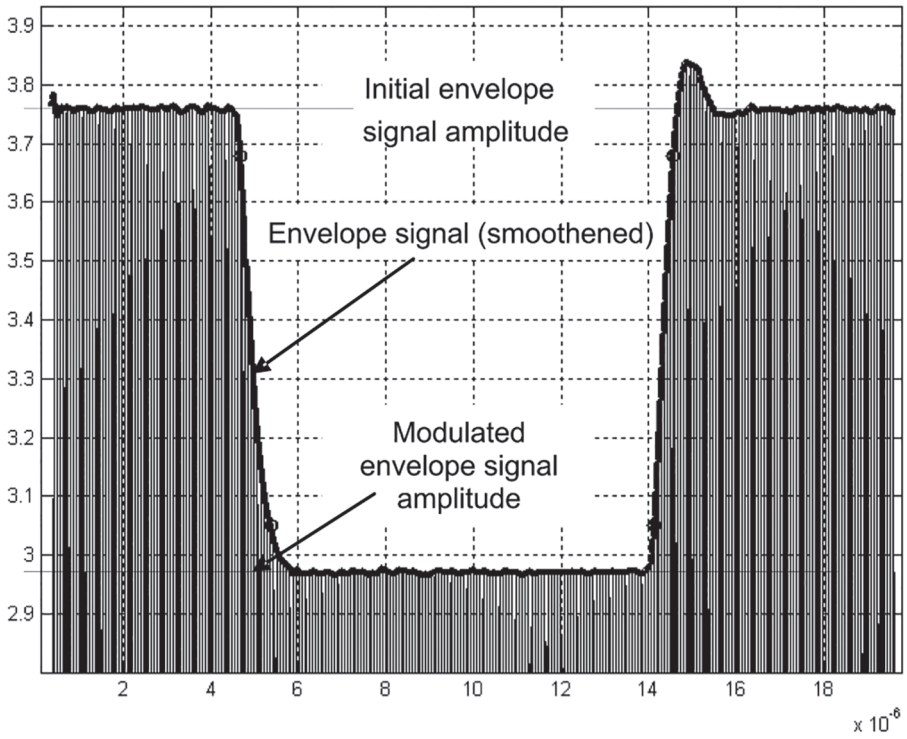


Figure E.5 — Envelope smoothing

E.5.2 Envelope smoothing for PCD to PICC bit rates of $f_c/8, f_c/4$ and $f_c/2$

No smoothing of the signal envelope shall be applied.

E.6 Modulation index *m* determination

The initial and modulated envelope signal amplitudes shall be determined by calculating the histogram of the smoothened envelope signal. The most frequent values correspond to the initial and modulated envelope signal amplitude. For Type A modulation signals only the initial envelope signal amplitude shall be determined using the histogram approach.

For PCD to PICC bit rates of $f_c/8$, $f_c/4$ and $f_c/2$ the minimum value of modulation index *m* shall be determined within the complete PCD frame (see Figure E.6). The PCD frame shall contain (10101010)b.

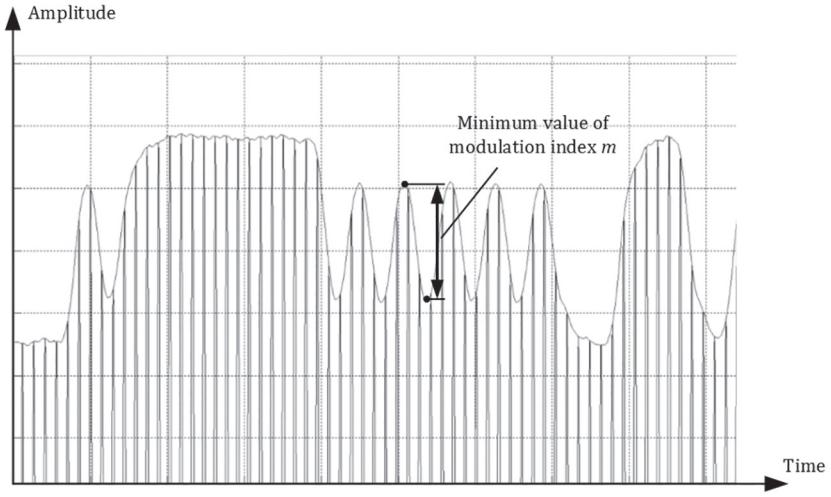


Figure E.6 — Minimum value of modulation index *m*

E.7 Timing determination

The rise and fall times should be determined according to the definitions in ISO/IEC 14443-2.

For PCD to PICC bit rates of $f_c/8$, $f_c/4$ and $f_c/2$ the timings shall be determined at positions with long modulation pulse positions, e.g., t_f at transition to SOF low and t_r at transition to EOF high.

E.8 Overshoot and undershoot determination

The envelope signal shall be smoothened by a moving average filter over 3 carrier periods before determining the overshoot and undershoot values according to the definitions in ISO/IEC 14443-2.

E.9 Program of the PCD modulation index *m* and waveform analysis tool

E.9.1 General

The following program written in C language gives an example for the implementation of the PCD modulation index *m* and waveform analysis tool.

The C program consists of 6 different files which should be placed in the same folder.

E.9.2 structures.h

```
/*********************************************************************/
/* structures.h                                                      */
/* This code contains the structures to save important results       */
/*********************************************************************/

#ifndef STRUCTURES_H
#define STRUCTURES_H

typedef struct esl
{
    double volt;
    double time;
    struct esl *sig;
}ESL;

typedef struct times
```

```
{
    double tf;           // Type B
    double tr;           // Type B
    double b;            // Type B
    double bVHBR;        // Type B VHBR
    double trstartind;   // Type B
    double trendind;     // Type B
    double tfstartind;   // Type B
    double tfendind;     // Type B

    double t1;           // Type A (all bit rates)
    double t1startind;   // Type A (all bit rates)
    double t1start;      // Type A (all bit rates)
    double t1endind;     // Type A (all bit rates)
    double t1end;        // Type A (all bit rates)

    double t2;           // Type A
    double t2startind;   // Type A
    double t2start;      // Type A
    double t3;           // Type A
    double t3end;        // Type A
    double t3endind;     // Type A
    double t4;           // Type A
    double t4endind;     // Type A

    double t5;           // Type A (higher bit rates)
    double t5startind;   // Type A (higher bit rates)
    double t6;           // Type A (higher bit rates)
    double t6end;        // Type A (higher bit rates)
    double t6endind;     // Type A (higher bit rates)
    double a;            // Type A 106-848
    double a_min;        // Type A 106
    double a_max;        // Type A 106
    double tploone;      // Type A (higher bit rates) /
                         // Type A 106 max ampl. during t2

    double tfmax;        // Type VHBR (higher bit rates)
    double tfmin;        // Type VHBR (higher bit rates)
    double tf_counter;   // Type VHBR (higher bit rates)
    double trmax;        // Type VHBR (higher bit rates)
    double trmin;        // Type VHBR (higher bit rates)
    double tr_counter;   // Type VHBR (higher bit rates)

    double mono;         // For monotonicity check and data (A & B)
}TIMES;

typedef struct shootreader
{
    double shootind;
    double shootind_b;
    double hf_reader;
    double hr_reader;
    double above;
    double above_b;
}SHOOTREADER;

#endif
```

E.9.3 fftrm.h

```
/*************************************************************************/
/*fftrm.h                                                                */
/*This is the header file for fftrm.c                                    */
/*************************************************************************/

#ifndef FFTRM_H
#define FFTRM_H

#define RE(z) ((z).r)
#define IM(z) ((z).i)
```

```
typedef float real;
typedef double doublereal;
typedef struct { real r, i; } complex;
typedef struct { doublereal r, i; } doublecomplex;

int zffts (int debug, doublecomplex *X, int M);
int ziffts (int debug, doublecomplex *X, int M);
void zfftrmc(doublecomplex *X, int M, int P, float D);
void rmpo (int *rv, int *rvp );

#endif
```

E.9.4 hilbert.h

```
/*********************************************************************/
/* hilbert.h                                                         */
/* This code contains the necessary functions for extracting envelope   */
/*********************************************************************/

#ifndef HILBERT_H_
#define HILBERT_H_

/* This function reads the sampled data recorded in the file */
int ReadData(void);

/* This function function performs the fourier transform */
void Fft(void);

/* This function performs the necessary phase shift */
void PhaseShifting(void);

/* This function performs the inverse fourier transfor */
void Ifft(void);

/* Envelope reconstruction is done by this function */
int EnvelopeReconstruction(void);

/* Hilbert main function */
void hilbert(char *fnamep,int size,double *voutput[],double *toutput[],int debugger);

#endif /* HILBERT_H_ */
```

E.9.5 hilbert.c

```
/*********************************************************************/
/* hilbert.c                                                         */
/* This program extracts the envelope of modulated carrier           */
/*                                                                   */
/* Input                                                             */
/*    File in text format containing a table of two columns          */
/*    (time and test PCD output voltage vd)                          */
/*    Data format of input-file:                                     */
/*    One data-point per line,                                       */
/*    {time[seconds], sense-coil-voltage[volts])                     */
/*    Data-points shall be equidistant time                          */
/*    Minimum sampling rate: 100 MSamples/second                     */
/*                                                                   */
/*    Example for spreadsheet file (start in next line):             */
/*    (time) , (voltage )                                            */
/*    3.00000e-06,1.00                                               */
/*    3.00200e-06,1.01                                               */
/*                                                                   */
/* Run:                                                              */
/* hilbert Filename.txt                                              */
/* or                                                                */
/* hilbert (default file name input.txt)                             */
/*********************************************************************/

# include <stdio.h>
```

```
# include <math.h>
# include <malloc.h>
# include <ctype.h>
# include <string.h>
# include "fftrm.h"

#define MAX_POINT 200000
#ifndef M_PI
#define M_PI 3.14159265358979323846264338327 95
#endif

int debug=0;
int fftdebug=0;

double *Gvalue;
double *Gtime;
double *Gr;
double *Gi;
double *Gc;
doublecomplex *Gt_ifft;

/* File containing the input data */
char *InputFileName ="input.txt";

int SampledPoints=0;
int N;
int row;
const int col=2;

int ReadData(void)
{
      float a,b;
      int i=0;
      FILE *fp1;
      i=0;
      SampledPoints=0; /* IA */

      if ((fp1 = fopen(InputFileName,"r")) == NULL)
      {
            printf("Cannot open input file.\n");
            return 1;
      }

      while(!feof(fp1))
      {
            fscanf(fp1,"%e,%e\n", &a, &b);

            Gtime[SampledPoints] = a;
            Gvalue[SampledPoints] = b;
            SampledPoints++;
            if (SampledPoints>= MAX_POINT) break;
      }
      fclose(fp1);

      fp1=fopen("inputfile.txt","w");
      if (!fp1)
      {
            fprintf(stdout,"Can't write the sampled data in inputfile.txt. \n");
            return 1;
      }
      for(i=0; i<SampledPoints; i++)
      fprintf(fp1,"%e\n",Gvalue[i]);
      /*Gtime[i] has been omitted*/
      fclose(fp1);

      if(debug)
      {
            fp1=fopen("inputtime.txt","w");
            if (!fp1)
            {
                  fprintf(stdout,"Can't write the sampled data in inputtime.txt. \n");
```

```
                return 1;
            }
            for(i=0; i<SampledPoints; i++)
            fprintf(fp1,"%e\n",Gtime[i]);
            /* Gtime[i] has been omitted */
            fclose(fp1);
      }

      if(debug)
      {
            if((fp1=fopen("inputfile.bin","wb"))!=NULL)
            {
                  fwrite(Gvalue,sizeof(double),SampledPoints,fp1);
                  fclose(fp1);
            }
      }

      if(SampledPoints<N)
      {
            for(i=SampledPoints;i<=N;i++)
            {
                  Gvalue[i] = 0;
            }
      }
      return 0;
}/*End Of Function ReadData;*/


void Fft(void)
{
      doublecomplex *Gt_freq;

      FILE *fp1,*fp2,*fp3;
      int k,num1,num2;

      Gt_freq = (doublecomplex *)calloc(sizeof(doublecomplex),row);

      /* FFT Procedure Starts for Sampled Data*/
      for(k=0;k<=N;k++)
      {
            RE(Gt_freq[k])=Gvalue[k];
            IM(Gt_freq[k])=0.0;
      }

      if(debug)
      {
            if((fp3=fopen("f.bin","wb"))!=NULL)
            {
                  fwrite(Gvalue,sizeof(double),row,fp3);
                  fclose(fp3);
            }
      }

      zffts(fftdebug,Gt_freq,row);
      /*FFT is done in spatial coordinate*/
      for (k=0;k<=N;k++)
      {
            Gr[k]=RE(Gt_freq[k]);
            Gi[k]=IM(Gt_freq[k]);
      }
      /* FFT Procedure Ends for Sampled Data */

      /* Writing The Real And Imaginary Part Of Reflected */
      /* Part for Debuging */
      /* Writing the real part of sampled data */

      if(debug)
      {
            if((fp1=fopen("Gr.bin","wb"))!=NULL)
            {
                  num1=fwrite(Gr,sizeof(double),row,fp1);
```

```
                fclose(fp1);
            }
            else
                fprintf(stdout,"Can't Open Gr.bin");

            /* Writing the img part of sampled data */
            if((fp2=fopen("Gi.bin","wb"))!=NULL)
            {
                num2=fwrite(Gi,sizeof(double),row,fp2);
                fclose(fp2);
            }
            else
                fprintf(stdout,"Can't Open Gi.bin");

            fprintf(stdout,"Num of Real Part Data after FFT = %d\n",num1);
            fprintf(stdout,"Num of Img Part Data after FFT = %d\n",num2);
        }

        free(Gt_freq);

}/* End Of The Function Fft */


void PhaseShifting(void)
{
        double *tempr, *tempi;
        int k;
        FILE *fp1;

        tempr = (double *)calloc(sizeof(double),row);
        tempi = (double *)calloc(sizeof(double),row);

        for ( k=0; k<=N; k++ )
        {
            tempr[k]=Gr[k];
            tempi[k]=Gi[k];
        }

        for ( k=0; k<=ceil(N/2); k++ )
        {
            Gr[k] = tempi[k];
            Gi[k] = -tempr[k];
        }

        for ( k=(int)ceil(N/2)+1; k<=N; k++ )
        {
            Gr[k] = -tempi[k];
            Gi[k] = tempr[k];
        }

        if(debug)
        {
            if((fp1=fopen("ffrpt.bin","wb"))!=NULL)
            {
                fwrite(Gr,sizeof(double),row,fp1);
                fclose(fp1);
            }
            if((fp1=fopen("ffipt.bin","wb"))!=NULL)
            {
                fwrite(Gi,sizeof(double),row,fp1);
                fclose(fp1);
            }
        }
        free (tempr);
        free (tempi);
}/*End of PhaseShift() function*/

void Ifft(void)
{
        double *Gt_tmp; /* It takes the real part of R_ifft*/
        double *Gt_tmpi;
```

```
	FILE *fp1;
	int k,i;

	Gt_tmp = (double *)calloc(sizeof(double),row);
	Gt_tmpi = (double *)calloc(sizeof(double),row);

	for (k=0;k<=N;k++)
	{
		Gt_ifft[k].r=Gr[k];
		Gt_ifft[k].i=Gi[k];
	}

	ziffts(fftdebug,Gt_ifft,row);/*IFFT of the signal in spatial coordinate*/

	/* End of IFFT */
	for (k=0;k<=N;k++)
	{
		Gt_tmp[k]=Gt_ifft[k].r;
	}

	if(debug)
	{
		fp1=fopen("ifft.txt","w");
		if (!fp1)
			fprintf(stdout,"Can't write in file");
		for(i=0; i<=N; i++)
			fprintf(fp1,"%.4e\n",(Gt_ifft[i].r));
		fclose(fp1);
	}

	if(debug)
	{
		if((fp1=fopen("iffrpt.bin","wb"))!=NULL)
		{
			fwrite(Gt_tmp,sizeof(double),row,fp1);
			fclose(fp1);
		}
		if((fp1=fopen("iffipt.bin","wb"))!=NULL)
		{
			fwrite(Gt_tmpi,sizeof(double),row,fp1);
			fclose(fp1);
		}
	}
	free(Gt_tmp );
	free(Gt_tmpi );
}/* End Of function Ifft*/

int EnvelopeReconstruction(void)
{
	FILE *fp1;
	int k;

	doublecomplex *G;
/* Input signal read from input file in complex form */
	doublecomplex *Ganalytical;
/* Analytical function of our input signal */

	double *test;
	double *sqrtr;
	double *sqrti;

	G = (doublecomplex *)calloc(sizeof(doublecomplex),row);
	Ganalytical = (doublecomplex *)calloc(sizeof(doublecomplex),row);

	test = (double *)calloc(sizeof(double),row);
	sqrtr=(double *)calloc(sizeof(double),row);
	sqrti=(double *)calloc(sizeof(double),row);

	for (k=0;k<=N;k++)
	{
		RE(G[k]) = Gvalue[k];
```

```
            IM(G[k]) = 0.0;
      }

      for (k=0;k<=N;k++)
      {
            RE(Ganalytical[k])=G[k].r;
            IM(Ganalytical[k])=Gt_ifft[k].r;
      }

      for (k=0;k<=N;k++)
      {
            sqrtr[k]=sqrt(Ganalytical[k].r*Ganalytical[k].
r+Ganalytical[k].i*Ganalytical[k].i);
      }

      if(debug)
      {
            fp1=fopen("output.txt","w");
            if (!fp1)
            {
                  fprintf(stdout,"Can't write extracted envelope in output.txt.\n");
                  free(G);
                  free(Ganalytical);
                  free(test);
                  free(sqrtr);
                  free(sqrti);
                  return 1;
            }

            for(k=0; k<SampledPoints; k++)
            {
                  fprintf(fp1,"%e,%e\n",Gtime[k],sqrtr[k]);
            }
            fclose(fp1);
      }
      else
      {
            memcpy(Gvalue,sqrtr,SampledPoints*(sizeof(double)));
      }
      free(G);
      free(Ganalytical);
      free(test);
      free(sqrtr);
      free(sqrti);
      return 0;
}

/* Hilbert function */
void hilbert(char *fnamep,int size,double *voutput[],double *toutput[],int debugger)
/* fnamep not needed */
{
      int status=0,i=1,k=0;
      char fname[256];
      strcpy(fname, fnamep);
      InputFileName= fname;
      SampledPoints=size;
      /* Reading the sampled data */
      debug = debugger;
      do
      {
            N=(int)pow(2,i)-1;
            i++;
      }while (MAX_POINT > N);

      if (debug)
            printf("N= %d\n",N);

      row=N+1;

      Gvalue = (double *)calloc(sizeof(double),row);
      Gtime = (double *)calloc(sizeof(double),row);
```

```
	Gr = (double *)calloc(sizeof(double),row);
	Gi = (double *)calloc(sizeof(double),row);
	Gt_ifft = (doublecomplex *)calloc(sizeof(doublecomplex),row);
	Gc = (double *)calloc(sizeof(double),row);

	/*for(k=0;k<size;k++)
	{
		Gvalue[k] = values[k];
		Gtime[k] = zeit[k];
	}*/
	if(debug)
	{
		status = ReadData();
		if (status== 1) goto MainExit;
	}
	else
	{
		memcpy(Gvalue,voutput,size*(sizeof(double)));
		memcpy(Gtime,toutput,size*(sizeof(double)));
	}

	/* Does FFT */
	Fft();

	/* Appropriate phase has been shifted */
	PhaseShifting();

	/*Does IFFT*/
	Ifft();

	/* Envelope reconstruction */
	status = EnvelopeReconstruction();
	if (status== 1) goto MainExit;

	if(!debug)
	{
		memcpy(voutput,Gvalue,size*(sizeof(double)));
		memcpy(toutput,Gtime,size*(sizeof(double)));
	}

	MainExit:
	free(Gvalue);
	free(Gtime);
	free(Gr);
	free(Gi);
	free(Gt_ifft);
	free(Gc);

}/* End of Hilbert function */
```

E.9.6 functs.c

```
/*************************************************************************/
/* functs.c                                                              */
/* This code contains the bigger part of the code: all functions which   */
/* provide the program functionality.                                    */
/* Main function of the whole program can be found at the end            */
/*************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <math.h>
#include "structures.h"
#include "hilbert.h"

#define MAX_SAMPLES 400000
#ifndef M_PI
#define M_PI 3.14159265358979323846264338327 95
```

```
#endif

int debugger=0;
char c;


// Reads a line from a file (f) and returns two char arrays (s and t)
// It is used to read files written in "comma separation" format.
void read_line (FILE *f, char *s, char *t)
{
    int a=0;
    int i=0;

    a=fgetc(f); /*Takes chars from the file pointed by f*/
    while (isspace(a)) /*spaces at the beginning of line are taken out*/
    {
        a=fgetc(f);
    }
    while (a!=',' && a!=EOF)
    {
        t[i++]=(char)a;
        a=fgetc(f);
    }
    t[i]='\0';
    i=0;
    a=fgetc(f);
    while (a!='\n' && a!=EOF)
    {
        s[i++]=(char)a;
        a=fgetc(f);
    }
    s[i]='\0'; /* We add the line end */
}

// Reads a line from a file (f) and discards it.
void skip_line (FILE *f)
{
    int a=0;

    a=fgetc(f);
    while (a!='\n' && a!=EOF)
    {
        a=fgetc(f);
    }
}

// Creates a ESL node with the given volt and time parameters
ESL *createnodef(double voltf, double timef)
{
    ESL *new=NULL;
    new=(ESL *)malloc(sizeof(ESL));
    if (new!=NULL)
    {
        new->volt=voltf;
        new->time=timef;
        new->sig=NULL;
    }
    else
        fprintf(stderr,"Memory Error");
    return new;
}

// Frees the allocated memory for ESL nodes
void freelist(ESL *first) // frees ESL list
{
    ESL *to_free;
    while (first->sig!=NULL)
    {
        to_free=first;
        first=first->sig;
        free(to_free);
```

```
    }
    free(first);
}

// Creates a TIME node with the given volt and time parameters
void createtime(TIMES *new, double tr, double tf, double b, double bVHBR, double
trstartind,
        double trendind, double tfstartind, double tfendind, double t1, double t1startind,
        double t1start, double t1endind, double t1end, double t2, double t2startind,
double t2start,
        double t3, double t3end, double t3endind, double t4, double t4endind, double t5,
        double t5startind, double t6, double t6end, double t6endind, double a, double a_
min, double a_max,
        double tploone,    double trmin, double trmax, double tr_counter, double tfmin,
double tfmax,
        double tf_counter, double mono_data)
{
    new->tf=tf;
    new->tr=tr;
    new->b=b;
    new->bVHBR=bVHBR;
    new->trstartind=trstartind;
    new->trendind=trendind;
    new->tfstartind=tfstartind;
    new->tfendind=tfendind;
    new->t1=t1;
    new->t1startind=t1startind;
    new->t1start=t1start;
    new->t1endind=t1endind;
    new->t1end=t1end;
    new->t2=t2;
    new->t2startind=t2startind;
    new->t2start=t2start;
    new->t3=t3;
    new->t3end=t3end;
    new->t3endind=t3endind;
    new->t4=t4;
    new->t4endind=t4endind;
    new->t5=t5;
    new->t5startind=t5startind;
    new->t6=t6;
    new->t6end=t6end;
    new->t6endind=t6endind;
    new->a=a;
    new->a_min=a_min;
    new->a_max=a_max;
    new->tploone=tploone;
    new->trmax=trmax;
    new->trmin=trmin;
    new->tr_counter=tr_counter;
    new->tfmax=tfmax;
    new->tfmin=tfmin;
    new->tf_counter=tf_counter;
    new->mono=mono_data;
}

// Inserts a ESL node (new) in a list pointed by "first"
void insert_node(ESL **first, ESL *new)
{
    ESL *p=NULL;
    ESL *previous=NULL;
    if (new!=NULL)
    {
        p=*first;
        if (p==NULL)
        {
            *first=new;
        }
        else
        {
            while (p!=NULL)
```

```
            {
                previous=p;
                p=p->sig;
            }
            previous->sig=new;
        }
    }
}

/* Multiplies order polynomials Supposing (x^2 + b*x + c) */
/* b and c are complex values stored in a table where even elements */
/* are real and odd elements imaginary */
double *mult_poli (int num_pol, double *b, double *c)
{
    int i=0;
    int y=0;
    double *real;
    double *imag;
    double *vector;
    double *new_real;
    double *new_imag;
    double real_b=0;
    double real_c=0;
    double imag_b=0;
    double imag_c=0;

    real=(double *)calloc(4*num_pol, sizeof(double));
    imag=(double *)calloc(4*num_pol, sizeof(double));
    new_real=(double *)calloc(4*num_pol, sizeof(double));
    new_imag=(double *)calloc(4*num_pol, sizeof(double));
    vector=(double *)calloc(4*num_pol, sizeof(double));

    real[0]=c[0];
    real[1]=b[0];
    real[2]=1;
    imag[0]=c[1];
    imag[1]=b[1];
    imag[2]=0;

    for (i=3; i<(4*num_pol); i++)
    {
        real[i]=0;
        imag[i]=0;
    }

    for (y=1; y<num_pol; y++)
    {
        // Selects values b and c
        real_b=b[2*y];
        real_c=c[2*y];
        imag_b=b[2*y+1];
        imag_c=c[2*y+1];

        for (i=0; i<=(2*num_pol-2); i++)
        {
            // Starts with coeff "c"
            new_real[i]+=real[i]*real_c-imag[i]*imag_c;
            new_imag[i]+=real[i]*imag_c+imag[i]*real_c;
            // Continues with coeff "b"
            new_real[i+1]+=real[i]*real_b-imag[i]*imag_b;
            new_imag[i+1]+=real[i]*imag_b+imag[i]*real_b;
            // Finishes with coeff "1"
            new_real[i+2]+=real[i];
            new_imag[i+2]+=imag[i];
        }

        // Update Values
        for (i=0;i<(4*num_pol); i++)
        {
            real[i]=new_real[i];
            imag[i]=new_imag[i];
```

```
            new_real[i]=0;
            new_imag[i]=0;
        }
    }

    for (y=0; y<(2*num_pol); y++)
    {
        vector[2*y]=imag[y];
        vector[2*y+1]=real[y];
    }

    free (new_imag);
    free (new_real);
    free (imag);
    free (real);

    return (vector);
}

// Part of the calculation of the butterworth coeffs.
double *butter_d_coeffs(double freq1, double freq2)
{
    int butter_order=2;
    int index=0;
    double theta=0;     // M_PI *(freq2-freq1)/2.0
    double cp=0;        // cosine of phi
    double *vec_r=0;    // z^-2 coefficients
    double *vec_t=0;    // z^-1 coefficients
    double *dcoeff=0;   // d coefficients
    double pole_ang=0;  // pole angle
    double divisor=0;

    cp = cos(M_PI*(freq2+freq1)/2.0);
    theta = M_PI*(freq2-freq1)/2.0;

    vec_r=(double *)calloc(2*butter_order,sizeof(double));
    vec_t=(double *)calloc(2*butter_order,sizeof(double));

    for(index=0;index<butter_order;++index)
    {
        pole_ang=M_PI*(double)(2*index+1)/(double)(2*butter_order);
        divisor=sin(2*theta)*sin(pole_ang)+1.0;
        vec_r[2*index]=cos(2*theta)/divisor;
        vec_r[2*index+1]=sin(2*theta)*cos(pole_ang)/divisor;
        vec_t[2*index]=-2.0*cp*(cos(theta)+sin(theta)*sin(pole_ang))/divisor;
        vec_t[2*index+1]=-2.0*cp*sin(theta)*cos(pole_ang)/divisor;
    }

    dcoeff=mult_poli(butter_order,vec_t,vec_r);

    dcoeff[4]=dcoeff[1];
//    dcoeff[3]=dcoeff[3];
    dcoeff[2]=dcoeff[5];
    dcoeff[1]=dcoeff[7];
    dcoeff[0]=1;

    for(index=5;index<=2*butter_order;index++)
        dcoeff[index]=0;

    free(vec_t);
    free(vec_r);

    return(dcoeff);
}

// Calculates the butterworth filter coefficients
void butterworth_coeffs(double freq1, double freq2, double *dfiltercoeff, double
*cfiltercoeff)
{
    // n filter order
    // freq1, freq2 lower/uppercutoff frequencies
```

```
    double sf;          // scaling factor
    double *dcoeff;     // d coefficients
    double cotan=0;     // cotangent of theta

    /* calculate the d coefficients */
    dcoeff=butter_d_coeffs(freq1,freq2);

    /* d coefficients for 4th order butterworth */
    dfiltercoeff[0]=dcoeff[0];  // Always 1
    dfiltercoeff[1]=dcoeff[1];
    dfiltercoeff[2]=dcoeff[2];
    dfiltercoeff[3]=dcoeff[3];
    dfiltercoeff[4]=dcoeff[4];

    /* scalling factor for the c filter coefficients */
    /* (Butterworth 4th order */
    cotan=1.0/tan(M_PI*(freq2-freq1)/2.0);
    sf=(1.0/(((cotan+sqrt(2)/2)*(cotan+sqrt(2)/2))+1/2));

    /* c coefficients for 4th order butterworth*/
    cfiltercoeff[0]=1*sf;
    cfiltercoeff[1]=0*sf;
    cfiltercoeff[2]=-2*sf;
    cfiltercoeff[3]=0*sf;
    cfiltercoeff[4]=1*sf;

    free(dcoeff);
}

// Checks if the data input is adequate to our algorithms
int datacheck(int posval, int negval, int samplesp, double tlast,FILE *pointfile)
{
    double diffr=0.0;
    char timestr1[25];
    char timestr2[25];
    char voltstr[25];
    double timestrf1=0;
    double timestrf2=0;
    double cut_sample=0;
    double delta_t=0;
    double val_t=0;
    int loop=0;
    double linf=0;
    int lind=0;

    // Checks that there are (nearly) much positive as negative values
    if (posval>negval)
        diffr=(posval-negval)/((posval+negval)/2);
    else
        diffr=(negval-posval)/((posval+negval)/2);

    if (diffr>0.8)
        fprintf(stdout,"Data Corrupted: Too little negative (or positive) values\n");

    // L=n*p with P=2*pi and n=1,2,3... - Cuts data
    rewind (pointfile);
    read_line(pointfile,voltstr,timestr1);
    read_line(pointfile,voltstr,timestr1);
    read_line(pointfile,voltstr,timestr1);
    // Skips csv header if present or not
    read_line(pointfile,voltstr,timestr1);
    while (voltstr[0]!='\0')
    {
        read_line(pointfile,voltstr,timestr2);
        loop++;
    }
    loop=loop+3;

    rewind(pointfile);
    for (lind=0; lind<loop; lind++)
```

```
    {
        read_line(pointfile,voltstr,timestr2);
    }

    timestrf1=atof(timestr1);            // t(4)
    timestrf2=atof(timestr2);            // t(end)
    cut_sample=((1/13.56e6)/((timestrf2-timestrf1)/(loop-1)));
    linf=samplesp;
    while (linf>0)
    {
        linf=linf-cut_sample;
    }
    linf+=cut_sample;
    samplesp=samplesp-linf-3;

    // At least 7 points per sample
    delta_t=tlast-timestrf1;
    val_t=delta_t/samplesp;
    if (val_t>(1/13.56e6)/7)
        fprintf(stdout, "More samples points needed - Nyquist\n");

    return (samplesp);
}

// Finds the most frequent value(s) of the given signal,
// Hmax (types A/B) and Hmin (type B)
void Hmaxfinder(double *env, double *Hmax, double *Hmin, int numsamples)
{
    int hist[2001]={0}; // IA changed memory access violation, incr +1
    int hi_low_i=0;
    double tophist=0;
    double bothist=100;
    double diffhist=0;
    double value=0;
    int histind=0;
    int max_i=0;
    int min_i=0;
    double max=0;
    double min=0;

    // Finds higher and lower values of samples
    for (hi_low_i=0; hi_low_i<MAX_SAMPLES; hi_low_i++)
    {
        if (env[hi_low_i]!=0)
        {
            if (env[hi_low_i]<bothist)
                bothist=env[hi_low_i];
            if (env[hi_low_i]>tophist)
                tophist=env[hi_low_i];
                // Finds limits for the histogram
        }
    }
    diffhist=tophist-bothist;

    for (hi_low_i=0; hi_low_i<numsamples; hi_low_i++)
    {
        if (env[hi_low_i]!=0)
        {
            value=env[hi_low_i];
            histind=(int)(2000*((value-bothist)/diffhist));
            // Performs a lineal quantization
            // IFX ==>US - One Cast is missing!!!
            hist[histind]++;
        }
    }

    for (hi_low_i=0; hi_low_i<1000; hi_low_i++)
    {
        if (hist[hi_low_i]>min)
        // Searchs most frequent value in the lower half of the form
        {
```

```
            min=hist[hi_low_i];
            min_i=hi_low_i;
        }
        *Hmin=(bothist+(diffhist/2000)*(min_i));
    }

    for (hi_low_i=1001; hi_low_i<2000; hi_low_i++)
    {
        if (hist[hi_low_i]>max)
        // Search most frequent value in the upper half of the form
        {
            max=hist[hi_low_i];
            max_i=hi_low_i;
        }
        *Hmax=(bothist+(diffhist/2000)*(max_i));
    }
}

// Finds the value of M_min for (type B - VHBR)
void Mminfinder(double *env, double Hmax, double Hmin, double *HmaxVHBR, TIMES *timeres,
int numsamples)
{
    int i=0;
    int j=0;
    double compare_hi=0.0;
    double compare_lo=0.0;
    double compare=0.0;
    double difference=0.0;
    int going_up=0;
    double ampl=0.0;
    double ampl_max=0.0;
    // represents the amplitude (Hmax-b), and indirectly "m"
    double m_deviation=0.0;
    // countermeasure 1: m_min < 0.2*m is not considered
    double Hmax_cm=0.0;
    // countermeasure 2: m_min does not start or end on borders
    double b_cm=0.0;
    double mmin=0.0;
    double mmin_cum=0.0;

    // Skip all zeros
    while (env[j]==0)
        j++;

    // where do we start?
    difference=env[j]-env[j+1];
    if (difference<0)
    {
        going_up=1; // going up
        compare_lo=env[j];
    }
    else if (difference>0)
    {
        going_up=0; // going down
        compare_hi=env[j];
    }
    compare=env[j];

    ampl_max=(Hmax-Hmin);
    m_deviation=ampl_max*0.4;
    Hmax_cm=Hmax*0.98;
    b_cm=Hmin*1.02;
    timeres->bVHBR=0;

    for (i=j; i<=numsamples-j; i++)
    {
        if (going_up==0) // GOING DOWN
        {
            if (compare>=env[i])
            {
                compare=env[i];
```

```
            }
            else if (compare<env[i])
            {
                compare=env[i];
                compare_lo=env[i];
                going_up=1; // change direction
                ampl=(compare_hi-compare_lo);
                mmin=(ampl/(compare_hi+compare_lo))*100;
                if (ampl>m_deviation && ampl<ampl_max && (compare_hi<Hmax_cm || compare_
lo>b_cm))
                //Countermeasures
                {
                    *HmaxVHBR=compare_hi;
                    timeres->bVHBR=compare_lo;
                    ampl_max=ampl;
                }
            }
        }
        if (going_up==1) // GOING UP
        {
            if (compare<=env[i])
            {
                compare=env[i];
            }
            else if (compare>env[i])
            {
                compare=env[i];
                compare_hi=env[i];
                going_up=0; // change direction
                ampl=(compare_hi-compare_lo);
                mmin=(ampl/(compare_hi+compare_lo))*100;
                if (ampl>m_deviation && ampl<ampl_max && (compare_hi<Hmax_cm || compare_
lo>b_cm))
                //Countermeasures
                {
                    *HmaxVHBR=compare_hi;
                    timeres->bVHBR=compare_lo;
                    ampl_max=ampl;
                    mmin_cum=mmin;
                }
            }
        }
    }
    if (*HmaxVHBR==0 || timeres->bVHBR==0)
    // in case waveform has only two levels (typical 1M7) Mmin=M
    {
        *HmaxVHBR=Hmax;
        timeres->bVHBR=Hmin;
    }
}

// Linear convolution (z= x convolve y)
void LinearConvolution(double X[],double Y[], double Z[], int lenx, int leny)
{
    double *zptr,s,*xp,*yp;
    int lenz;
    int i,n,n_lo,n_hi;

    lenz=lenx+leny-1;
    zptr=Z;

    for (i=0;i<lenz;i++)
    {
        s=0.0;
        n_lo=0>(i-leny+1)?0:i-leny+1;
        n_hi=lenx-1<i?lenx-1:i;
        xp=X+n_lo;
        yp=Y+i-n_lo;

        for (n=n_lo;n<=n_hi;n++)
        {
```

```
            s+=*xp * *yp;
            xp++;
            yp--;
        }

        *zptr=s;
        zptr++;
    }
}

int envfilt(int rate, double *output, double *toutput, int filterlength, double tini,
double tend, int lengthp, double *envelope)
{
    int cofpi=0;
    int xx=0;
    double cofp=0.0;
    int lengthp1=0;
    double lengthf=0;
    double cof[2000]={0};
    double points=0.0;
    int pointsi=0;
    int lengthtotal=0;

    cofp=(73.75e-9)/((tend-tini)/(lengthp));
    cofpi=cofp+0.5;
    lengthf=cofpi*filterlength;
    points=(5*73.75e-9)/((tend-tini)/(lengthp-1));
    pointsi=(int)points+1;
    lengthp1=lengthp;

    for (xx=0; xx<lengthf; xx++)
        cof[xx]=1/lengthf;

    for (xx=lengthf+1; xx<2000; xx++)
        cof[xx]=0;

    if (rate==106 || rate==212 || rate==424 || rate==848)
    {
        LinearConvolution(cof, output, envelope, lengthf, lengthp);
    }

    else if (rate==1700 || rate==3400 || rate==6800)
    {
        cof[0]=1;
        for (xx=1; xx<2000; xx++)
            cof[xx]=0;
        lengthf=1;
        LinearConvolution(cof, output, envelope, lengthf, lengthp);
    }

    for (xx=0; xx<(pointsi); xx++) // "Cuts" envelope
    {
        envelope[xx]=0.0;
        toutput[xx]=0.0;
        envelope[lengthp1-xx]=0.0;
        toutput[lengthp1-xx]=0.0;
    }
    for (xx=lengthp1+1; xx<MAX_SAMPLES; xx++)
    {
        envelope[xx]=0.0;
        toutput[xx]=0.0;
    }

    lengthtotal=lengthp1-2*(pointsi);
    return (lengthtotal);
}

// Performs the search of a certain level (target) in the envelope,
// i.e. 5% ,60%, 90% in type A, 106 Kbps
int localizador(double *env, double *toutput, double target, ESL **crosses, int env_
length)
```

```
{
    int flag=0;
    double diff;
    ESL *new;
    double v;
    double t;
    int crosscounter=0;
    int locat_index=0;
    int locat_index_start=0;

    while (env[locat_index]==0.0) // Leaves 0s out
        locat_index++;

    locat_index_start=locat_index;
    if (env[locat_index]-target>0)
    {
        for (locat_index=locat_index_start; locat_index<env_length+locat_index_start-1;
locat_index++) // IFX
        {
            diff=env[locat_index]-target;
            if (diff<0 && flag==0 && env[locat_index]!=0.0)
            // At the beginning or after an odd occurrence, envelope
            // is over "target" level
            {
                flag=1; // down!
                v=target;
                t=toutput[locat_index-1]+((toutput[locat_index]-toutput[locat_index-1])/
(env[locat_index]-env[locat_index-1])*(target-env[locat_index-1]));
                new=createnodef(v,t);
                insert_node(crosses,new);
                crosscounter++;
            }
            if (diff>0 && flag==1 && env[locat_index]!=0.0)
// After first (or even) occurrence, envelope is under "target" level
            {
                flag=0; // up!
                v=target;
                t=toutput[locat_index-1]+((toutput[locat_index]-toutput[locat_index-1])/
(env[locat_index]-env[locat_index-1])*(target-env[locat_index-1]));
                new=createnodef(v,t);
                insert_node(crosses,new);
                crosscounter++;
            }
        }
// Returns all occurrences with time and volt level in a list
    }
    else
        fprintf(stdout,"Signal is not ---|___|--- \n");

    return (crosscounter);
// Also returns how many occurrences appeared
}

// Function that calculates the relevant times
void tfinder(char type, double *env, double *toutput, double tini, double Hmax, double
Hmin, int rate, int env_length, TIMES *timeres)
{
    double *envc=NULL;
    double *envc2=NULL;
    double *toutput2=NULL;
    ESL *crosses=NULL;
    ESL *crosses2=NULL;
    ESL *crosses3=NULL;
    ESL *crosses_WORK=NULL;
    ESL *crosses_WORK2=NULL;
    double ninety=0.0;
    double five=0.0;
    double sixty=0.0;
    double tp90one=0.0;
    double tp90two=0.0;
    double tp5one=0.0;
```

```
double tp5two=0.0;
double tp60two=0.0;
double vp90one=0.0;
double vp90two=0.0;
double vp5one=0.0;
double vp5two=0.0;
double vp60two=0.0;
double tphione=0.0;
double tphitwo=0.0;
double vphione=0.0;
double vphitwo=0.0;
double tpmidone=0.0;
double vpmidone=0.0;
double tploone=0.0;
double tplotwo=0.0;
double vplotwo=0.0;
double t1=0.0;
double t2=0.0;
double t3=0.0;
double t4=0.0;
double t5=0.0;
double t6=0.0;
int flag=0;
int flag2=0;
int flag3=0;
int flag_improv=0;
int x_improv=0;
double minvolt=0.0;
double highrate_low=0.0;
double highrate_mid=0.0;
double highrate_hi=0.0;
double a=0.0;  // Attention! a is measured in volts (not %)
double t6end=0.0;
double t5startind=0.0;
double t6endind=0.0;
double b=0.0;
double B_low=0.0;
double B_hi=0.0;
double tr=0.0;
double tf=0.0;
double tfstartind=0.0;
double tfendind=0.0;
double trstartind=0.0;
double trendind=0.0;
double t2startind=0.0;
double t2start=0.0;
double t1startind=0.0;
double t1start=0.0;
double t1end=0.0;
double t3end=0.0;
double t4endind=0.0;
double t3endind=0.0;
double t1endind=0.0;
int index_A=0;
int index_A2=0;
int residual_limit=0.0;
double max_A=0;
double min_A=0;
double avg_A=0;
double reference=0.0;

// Variables for VHBR
int counter=0;
int rev_counter=0;
int VHBR_step=0;
double VHBR_tr=0.0;
double VHBR_tf=0.0;
double tr_accum=0.0;
double tf_accum=0.0;
int tr_counter=0;
int tf_counter=0;
```

```
    double t_one_sample=0.0;
    double tlo=0.0;
    double vlo=0.0;
    double thi=0.0;
    double vhi=0.0;
    double trmin=100;
    double trmax=0.0;
    double tfmin=100;
    double tfmax=0.0;
    int mono=0;

    ESL *low_strt=NULL;
    ESL *hi_strt=NULL;

    b=Hmin;
    envc=env;
    envc2=envc;
    toutput2=toutput;

    switch (type)
    {
        case 'A':
        {
            switch (rate)
            {
                case 106:
                {
                    ninety=Hmax*0.9;
                    five=Hmax*0.05;
                    sixty=Hmax*0.6;

                    flag2=localizador(envc,toutput,five,&crosses2,env_length);
// Finds 5% of Hmax
                    if (flag2==2)
// if there are two occurrences, there´s no problem
                    {
                        tp5one=crosses2->time;
// Temporary values are stored for future use
                        vp5one=crosses2->volt;
                        tp5two=crosses2->sig->time;
                        vp5two=crosses2->sig->volt;
                        freelist(crosses2);
                    }
                    else if (flag2==0)
                    // ...if there is no occurrence...
                        fprintf(stdout,"5 percent of Hmax not reached - maybe wrong type
or bitrate? \n");

                    else if (flag2>2)
// ...if there are more than two occurrences...
                    {
                        tp5one=crosses2->time;
// Temporary values are stored for future use
                        vp5one=crosses2->volt;

                        while(crosses2->sig!=NULL)
                            crosses2=crosses2->sig;

                        tp5two=crosses2->time;
                        vp5two=crosses2->volt;
                        freelist(crosses2);
                        mono=-1;
                    }

                    flag=localizador(envc,toutput,ninety,&crosses,env_length);
// Finds 90% of Hmax
                    if (flag>=2)
                    {
                        crosses_WORK=crosses;
// Copy of crosses to work with
                        while (x_improv<flag)
```

```
                        {
                            if (crosses_WORK->time<tp5one)
                            {
                                tp90one=crosses_WORK->time;
// Temporary values are stored for future use
                                vp90one=crosses_WORK->volt;
                            }

                            if (crosses_WORK->time>tp5two && flag_improv==0)
                            {
                                tp90two=crosses_WORK->time;
// Temporary values are stored for future use
                                vp90two=crosses_WORK->volt;
                                flag_improv=1;
                            }
                                crosses_WORK=crosses_WORK->sig;
                                x_improv++;
                        }
                        freelist(crosses);
                    }

                    else // ...otherwise...
                    {
                        fprintf(stdout,"90 %% of Hmax not found - Noise Too High \n");
                    }

                    flag_improv=0;
                    x_improv=0;

flag3=localizador(envc,toutput,sixty,&crosses3,env_length);
// Finds 60% of Hmax

                    if (flag3>=2)
                    {
                        crosses_WORK2=crosses3;
// Copy of crosses to work with
                        while (x_improv<flag3)
                        {
                            if (crosses_WORK2->time>tp5two && flag_improv==0)
                            {
                                tp60two=crosses_WORK2->time;
// Temporary values are stored for future use
                                vp60two=crosses_WORK2->volt;
                                flag_improv=1;
                            }
                                crosses_WORK2=crosses_WORK2->sig;
                                x_improv++;
                        }
                        freelist(crosses3);
                    }

                    t1=tp5two-tp90one;
// Definitive values are calculated and stored for display
                    t2=tp5two-tp5one;
                    t3=tp90two-tp5two;
                    t4=tp60two-tp5two;

                    t1start=tp90one;
// Other important values for the coming functions
                    t1end=tp5two;
                    t2start=tp5one;
                    t3end=tp90two;
                    t1startind=vp90one;
                    t1endind=vp5two;
                    t2startind=vp5one;
                    t3endind=vp90two;
                    t4endind=vp60two;

// NEW 2016.07.19 CALCULATION OF "a" FOR 106 (MAX during t2)
                    index_A=0;
                    index_A2=0;
```

```
                    residual_limit=0;
                    max_A=0.0;
                    min_A=100.0;
                    avg_A=0.0;

                    while(toutput[index_A]==0 || toutput[index_A]<=t2start)
                    {
                        index_A++; // go to t2start
                    }
                    while(toutput[index_A2]==0 || toutput[index_A2]<=t1end)
                    {
                        index_A2++; // go to t2end
                    }
                    residual_limit=(index_A2-index_A)*0.6;
// so many samples include t2 * 60%
                    index_A=index_A2-residual_limit;
// move start to 60% of t2
                    reference=envc2[index_A2];
                    index_A2--;
                    while(envc2[index_A2]<reference)
// skip the "rest" of the rising edge
                    {
                        reference=envc2[index_A2];
                        index_A2--;
                    }

                    residual_limit=(index_A2-index_A);
// once removed initial part of rising edge
                    while(index_A<index_A2)
// here we look for "a"
                    {
                        reference=envc2[index_A];
                        avg_A+=reference;
//sum to calculate average
                        if(reference>max_A)
                            max_A=reference; //max a
                        if(reference<min_A)
                            min_A=reference; //min a
                        index_A++;
                    }

                    a=avg_A/residual_limit;
// WHERE TO PASS THIS DATA IN THE STRUCT?? This parameter is used for
// the residual carrier.
// END OF NEW 2016.07.19 CALCULATION OF "a" FOR 106 (MAX during t2)

                    createtime(timeres,0,0,0,0,0,0,0,0,t1,t1startind,t1start,t1endind,t1en
d,t2,t2startind,t2start,t3,t3end,t3endind,t4,t4endind,0,0,0,0,0,a,min_A,max_A,0,0,0,0,0,0,
0,mono);        // Residual Carrier VERIFIKATIONSTEAM
                }
                break;

                case 212:
                case 424:
                case 848:
                {
                    ninety =Hmax*0.9;
                    while (env[index_A]==0.0)
// Finds first value different of 0.0
                        index_A++;

                    minvolt=env[index_A];
                    while (env[index_A]!=0.0)
// All values are considered
                    {
                        if (env[index_A]<minvolt)
                        {
                            minvolt=env[index_A];
// Finds minimal voltage
                        }
                        index_A++;
```

```
                    }

                    highrate_low=minvolt+0.1*(Hmax-minvolt);
// Calculates target
                    flag2=localizador(envc,toutput,highrate_low,&crosses,env_length);
// Finds target
                    if (flag2==2)
// ...if there are two occurrences, there´s no problem...
                    {
                        tploone=crosses->time;
// Temporary values are stored for future use
                        tplotwo=crosses->sig->time;
                        vplotwo=crosses->sig->volt;
                    }

                    else if (flag2>2)
// if there are more than two occurrences...
                    {
                        tploone=crosses->time;
// Temporary values are stored for future use

                        while(crosses->sig!=NULL)
                            crosses=crosses->sig;

                        tplotwo=crosses->time;
                        vplotwo=crosses->volt;
                    }

                    highrate_hi=ninety+0.1*minvolt;
// Calculates target
                    flag=localizador(envc,toutput,highrate_hi,&crosses2,env_length);
// Finds target
                    if (flag>=2)
                    {
                        crosses_WORK=crosses2;
                        while (x_improv<flag)
                        {
                            if (crosses_WORK->time<tploone)
                            {
                                tphione=crosses_WORK->time;
// Temporary values are stored for future use
                                vphione=crosses_WORK->volt;
                            }

                            if (crosses_WORK->time>tplotwo && flag_improv==0)
                            {
                                tphitwo=crosses_WORK->time;
// Temporary values are stored for future use
                                vphitwo=crosses_WORK->volt;
                                flag_improv=1;
                            }
                                crosses_WORK=crosses_WORK->sig;
                                x_improv++;
                        }
                        freelist(crosses2);
                    }

                    Else // ...otherwise...
                    {
                        fprintf(stdout,"90 %% of Hmax not reached! \n");
                    }

                    x_improv=0;

                    highrate_mid=(Hmax+minvolt)/2;
// Calculates target
                    flag3=localizador(envc,toutput,highrate_mid,&crosses3,env_length);
// Finds target

                    if (flag3>=2)
                    {
```

```
                        crosses_WORK2=crosses3;
// Copy of crosses to work with
                        while (x_improv<flag3)
                        {
                            if (crosses_WORK2->time<tploone)
                            {
                                tpmidone=crosses_WORK2->time;
// Temporary values are stored for future use
                                vpmidone=crosses_WORK2->volt;
                            }
                            x_improv++;
                        }
                        freelist(crosses3);
                    }

                    else // ...otherwise...
                        fprintf(stdout,"Noise Too High \n");

                    t1=tplotwo-tphione;
// Definitive values are calculated and stored for display
                    t5=tplotwo-tpmidone;
                    t6=tphitwo-tplotwo;
                    a=minvolt;

                    t6end=tphitwo;
// Other important values for the coming functions
                    t1start=tphione;
                    t1startind=vphione;
                    t5startind=vpmidone;
                    t1endind=vplotwo;
                    t6endind=vphitwo;

                    createtime(timeres,0,0,0,0,0,0,0,0,t1,t1startind,t1start,t1endind,t1en
d,0,0,0,0,0,0,0,0,t5,t5startind,t6,t6end,t6endind,a,0,0,tploone,0,0,0,0,0,0,0);
                }
                break;
            }
        }
        break;

        case 'B':
        {
            switch (rate)
            {
                case 106:
                case 212:
                case 424:
                case 848:
                {
                    B_low=b+0.1*(Hmax-b);
// Calculates target
                    flag=localizador(envc,toutput,B_low,&crosses,env_length);
// Finds target
                    if (flag>=2)
                    {
                        crosses_WORK=crosses;
                        tploone=crosses_WORK->time;
// Temporary values are stored for future use
                        while (x_improv<flag)
                        {
                            tplotwo=crosses_WORK->time;
// Temporary values are stored for future use
                            vplotwo=crosses_WORK->volt;
                            crosses_WORK=crosses_WORK->sig;
                            x_improv++;
                        }
                        freelist(crosses);
                    }

                    B_hi=Hmax-0.1*(Hmax-b);
// Calculates target
```

```
                    flag=localizador(envc,toutput,B_hi,&crosses2,env_length);
// Finds target
                    if (flag>=2)
                    {
                        x_improv=0;
                        flag_improv=0;
                        crosses_WORK=crosses2;
                        while (x_improv<flag)
                        {
                            if (crosses_WORK->time<tploone)
                            {
                                tphione=crosses_WORK->time;
// Temporary values are stored for future use
                                vphione=crosses_WORK->volt;
                            }

                            if (crosses_WORK->time>tplotwo && flag_improv==0)
                            {
                                tphitwo=crosses_WORK->time;
// Temporary values are stored for future use
                                vphitwo=crosses_WORK->volt;
                                flag_improv=1;
                            }
                                crosses_WORK=crosses_WORK->sig;
                                x_improv++;
                        }
                        freelist(crosses2);
                    }

                    tf=tploone-tphione;
// Definitive values are calculated and stored for display
                    tr=tphitwo-tplotwo;
                    tfstartind=tphione;
// Other important values for the coming functions
                    tfendind=tploone;
                    trstartind=tplotwo;
                    trendind=tphitwo;

                    createtime(timeres,tr,tf,b,0,trstartind,trendind,tfstartind,tfend
ind,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);
                }
                break;

                case 1700:
                case 3400:
                case 6800:
                {
                    B_low=b+0.1*(Hmax-b);
// Calculates target
                    B_hi=Hmax-0.1*(Hmax-b);
// Calculates target

                    flag=localizador(envc,toutput,B_low,&crosses,env_length);
                    flag2=localizador(envc,toutput,B_hi,&crosses2,env_length);

                    tfstartind=crosses2->time;
// Reused as start point for overshoot and undershoot
                    tfendind=crosses->time;

                    low_strt=crosses;
                    hi_strt=crosses2;

                    // LOCATE ADJACENT POINTS
                    while (crosses->sig!=NULL && crosses2->sig!=NULL)
                    {
                        tlo=crosses->time;
                        thi=crosses2->time;
                        if (thi<tlo) // FALLING EDGE
                        {
                            if (crosses2->sig->time < tlo)
// Discard Point
```

```
                            crosses2=crosses2->sig;
                        else if (crosses2->sig->time > tlo)
// Analysis Tf
                        {
                            vlo=crosses->volt;
                            vhi=crosses2->volt;
                            while (toutput2[counter]==0)
// Set counters
                            {
                                counter++;
                                rev_counter++;
                            }
                            t_one_sample=toutput2[counter+2]-toutput2[counter+1];
                            while (toutput2[counter]<=thi)
// Set counters
                            {
                                counter++;
                                rev_counter++;
                            }
                            while (toutput2[rev_counter]<=tlo)
// Set counters
                                rev_counter++;

                            while (vlo<vhi)
                            {
                                vlo=envc2[rev_counter-VHBR_step];
                                vhi=envc2[counter+VHBR_step];
                                VHBR_step++;
                            }
                            if (vlo==vhi)
                                VHBR_step=VHBR_step*2;
                            else if (vlo>vhi)
                                VHBR_step=VHBR_step*2-1;

                            VHBR_tf=VHBR_step*t_one_sample;
                            tf_counter++;
                            tf_accum=tf_accum+VHBR_tf;

                            if (VHBR_tf>tfmax)
// NEW V2_59 EDIN
                                tfmax=VHBR_tf;
                            if (VHBR_tf<tfmin)
                                tfmin=VHBR_tf;

                            VHBR_step=0.0;
// Reset Counters
                            VHBR_tf=0.0;
                            counter=0;
                            rev_counter=0;
                            crosses2=crosses2->sig;
                        }
                    }

                    else if (tlo<thi)
// RISING EDGE
                    {
                        if (crosses->sig->time < thi)
// Discard Point
                            crosses=crosses->sig;
                        else if (crosses->sig->time > thi)
// Analysis Tr
                        {
                            vlo=crosses->volt;
                            vhi=crosses2->volt;
                            while (toutput2[counter]==0)
// Set counters
                            {
                                counter++;
                                rev_counter++;
                            }
                            t_one_sample=toutput2[counter+2]-toutput2[counter+1];
```

```
                                while (toutput2[counter]<=tlo)
// Set counters
                                {
                                    counter++;
                                    rev_counter++;
                                }
                                while (toutput2[rev_counter]<=thi)
// Set counters
                                    rev_counter++;

                                while (vlo<vhi)
                                {
                                    vhi=envc2[rev_counter-VHBR_step];
                                    vlo=envc2[counter+VHBR_step];
                                    VHBR_step++;
                                }
                                if (vlo==vhi)
                                    VHBR_step=VHBR_step*2;
                                else if (vlo>vhi)
                                    VHBR_step=VHBR_step*2-1;

                                VHBR_tr=VHBR_step*t_one_sample;
                                tr_counter++;
                                tr_accum=tr_accum+VHBR_tr;

                                if (VHBR_tr>trmax)
// NEW V2_59 EDIN
                                    trmax=VHBR_tr;
                                if (VHBR_tr<trmin)
                                    trmin=VHBR_tr;

                                VHBR_step=0.0;
// Reset counters
                                VHBR_tr=0.0;
                                counter=0;
                                rev_counter=0;
                                crosses=crosses->sig;
                            }
                        }
                    }
// Calculate and Save Params
                    tf=tf_accum/tf_counter;
// Definitive values are calculated and stored for display
                    tr=tr_accum/tr_counter;
// Reused as end point for overshoot
                    trendind=crosses2->time;
                    trstartind=crosses->time;

                    createtime(timeres,tr,tf,b,0,trstartind,trendind,tfstartind,tfendind,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,trmin,trmax,tr_counter,tfmin,tfmax,tf_
counter,0);
                }
                freelist(low_strt);
                freelist(hi_strt);
                break;

            }
        }
        break;
    }
}

// Checks monotony on the falling edge
void monocheck(double *env, double *toutput, double Hmax, TIMES *timesp, int rate, char
type)
{
    double tinit=0.0;
    double tend=0.0;
    double compare=0.0;
    double timer0=0.0;
    double timer1=0.0;
```

```
    double volt0=0.0;
    double volt1=0.0;
    int counter=0;
    double max_diff=0.0;
    int flag_mono=0;
    int counter_memo=0;
    double max_diff_max=0.0;

    flag_mono=timesp->mono;

    switch (type)
    {
        case 'A':
        {
            switch (rate)
            {
                case 106:
                {
                    while (env[counter]==0)
                        counter++;

                    tinit=timesp->t1start;
                    tend=timesp->t2start;

                    while (toutput[counter]<tinit)
                        counter++; //find first value

                    while (toutput[counter]<tend)
                    {
                        compare=env[counter];
                        if (compare<env[counter+1])
                        {
                            timer0=toutput[counter];
                            volt0=env[counter];
                            while (volt0<env[counter+1])
// growing values...
                            {
                                counter++;
                                volt0=env[counter];
                            }
                            counter_memo=counter;
                            timer1=toutput[counter];
// ...max.value -> timer1
                            volt1=env[counter];

                            while(env[counter-1]<volt1)
// go back until we find max value again
                            {
                                counter--;
                            }
                            timer0=toutput[counter--];
// max.value -> timer0
                            volt0=env[counter--];

                            max_diff=timer1-timer0;
                            counter=counter_memo;
                            if (max_diff_max<=max_diff)
// Max difference between peak and previous same value (time)
                            {
                                max_diff_max=max_diff;
                                timesp->mono=max_diff;
                                timesp->mono=timesp->mono*flag_mono;
                            }
                        }
                        else
                            counter++;
                    }

                }
                break;
```

```
                case 212:
                case 424:
                case 848:
                {
                    while (env[counter]==0)
                        counter++;

                    tinit=timesp->t1start;
                    tend=timesp->tploone;

                    while (toutput[counter]<tinit)
                        counter++; //find first value

                    while (toutput[counter]<tend)
                    {
                        compare=env[counter];
                        if (compare<env[counter+1])
                        {
                            timer0=toutput[counter];
                            volt0=env[counter];
                            volt1=volt0;
                            while (volt1<env[counter+1])
// growing values...
                            {
                                counter++;
                                volt1=env[counter];
// volt1 is the peak of the hump
                            }
                            timer1=toutput[counter];

                            max_diff=volt1-volt0;
// Max. Diff between local max and previous min (volt)

                            if (max_diff_max<max_diff)
                            {
                                max_diff_max=max_diff;

                                timesp->mono=max_diff/Hmax;
                            }
                        }
                        else
                            counter++;
                    }
                }
                break;
            }
        }
        break;

        case 'B':
        {
            while (env[counter]==0)
                counter++;

            tinit=timesp->tfstartind;
            tend=timesp->tfendind;
            while (toutput[counter]<tinit)
                counter++;
// find first value
            while (toutput[counter]<tend && timesp->mono==0)
            {
                compare=env[counter];
                if (compare<env[counter+1])
                    timesp->mono=-1;
// falling edge not monotonic
                counter++;
            }

            tinit=timesp->trstartind;
            tend=timesp->trendind;
            while (toutput[counter]<tinit)
```

```
                counter++;
// find first value

            while (toutput[counter]<tend && (timesp->mono==0 || timesp->mono==-1))
            {
                compare=env[counter];
                if (compare>env[counter+1])
                {
                    if (timesp->mono==0 || timesp->mono==-2)
                        timesp->mono=-2;
// rising edge not monotonic
                    else
                        timesp->mono=-3;
// falling and rising edges not monotonic
                }
                counter++;
            }
        }
        break;
    }
}

// Function that calculates the overshoot times
void overshoot(TIMES *timesp, double Hmax, double *env2, double *toutput, int rate, char
type, int samples, SHOOTREADER *shootreader)
{
    double shootind=0.0;
    double shootind_b=0.0;
    double hr_reader=0.0;
    double hf_reader=0.0;
    double above=0.0;
    double above_b=10.0;
    double start=0.0;
    int index_samples=0;

    switch (type)
    {
        case 'A':
        {
            switch (rate)
            {
                case (106):
                {
                    start=timesp->t3end;
                    while (toutput[index_samples]<=start)
                        index_samples++;

                    while (index_samples<=samples)
                    {
                        if (env2[index_samples]>above)
                        {
                            above=env2[index_samples];
                            shootind=toutput[index_samples];
                        }
                        index_samples++;
                    }

                    if (above<Hmax)
// In very strange cases if there´s no overshoot, the highest point
                        above=Hmax;
// In the curve can be cutted off by envfilt, producing a negative hr
                }
                break;
                case (212):
                case (424):
                case (848):
                {
                    start=timesp->t6end;
                    while (toutput[index_samples]<=start)
                        index_samples++;
```

```
                    while (index_samples<=samples)
                    {
                        if (env2[index_samples]>above)
                        {
                            above=env2[index_samples];
                            shootind=toutput[index_samples];
                        }
                        index_samples++;
                    }

                    if (above<Hmax)
// In very strange cases if there´s no overshoot, the highest point
                        above=Hmax;
// In the curve can be cutted off by envfilt, producing a negative hr
                }
                break;
            }
        }
        break;
        case 'B':
        {
            start=timesp->trendind;
            while (toutput[index_samples]==0)
                index_samples++;
            while (toutput[index_samples]<start)
// Starts at the rising edge
                index_samples++;

            while (env2[index_samples]!=0)
            {
                if (env2[index_samples]>above)
                {
                    above=env2[index_samples];
                    shootind=toutput[index_samples];
                    hr_reader=(above-Hmax)/(Hmax-timesp->b);
                    if (hr_reader<0)
// In very strange cases if there´s no overshoot, the highest point
                        hr_reader=0;
// In the curve can be cutted off by envfilt, producing a negative hr
                }
                index_samples++;
            }

            index_samples=0;
            start=timesp->tfendind;
            while (toutput[index_samples]==0)
                index_samples++;
            while (toutput[index_samples]<start)
                index_samples++;

            while (env2[index_samples]!=0)
            {
                if (env2[index_samples]<above_b)
                {
                    above_b=env2[index_samples];
                    shootind_b=toutput[index_samples];
                    hf_reader=(timesp->b-above_b)/(Hmax-timesp->b);
                }
                index_samples++;
            }
        }
        break;

    }
    shootreader->shootind=shootind;
    shootreader->shootind_b=shootind_b;
    shootreader->hr_reader=hr_reader;
    shootreader->hf_reader=hf_reader;
    shootreader->above=above;
    shootreader->above_b=above_b;
}
```

```
// Calculates the modulation index "m"
double modulation(char type, double Hmax, double b)
{
    double m=0;
    switch (type)
    {
        case 'A':
        {
            // m is not defined for type A
        }
        break;
        case 'B':
        {
            m=100*(Hmax-b)/(Hmax+b); // In %
        }
        break;
    }
    return (m);
}

char * dtoa(char *s, double n)
{
    double PRECISION = 0.00000000000001;
    // handle special cases
    if (isnan(n))
    {
        strcpy(s, "nan");
    }
    else if (isinf(n))
    {
        strcpy(s, "inf");
    }
    else if (n == 0.0)
    {
        strcpy(s, "0");
    }
    else
    {
        int digit, m, m1;
        char *c = s;
        int neg = (n < 0);
        if (neg)
            n = -n;
        // calculate magnitude
        m = log10(n);
        int useExp = (m >= 14 || (neg && m >= 9) || m <= -9);
        if (neg)
            *(c++) = '-';
        // set up for scientific notation
        if (useExp) {
            if (m < 0)
               m -= 1.0;
            n = n / pow(10.0, m);
            m1 = m;
            m = 0;
        }
        if (m < 1.0) {
            m = 0;
        }
        // convert the number
        while (n > PRECISION || m >= 0) {
            double weight = pow(10.0, m);
            if (weight > 0 && !isinf(weight)) {
                digit = floor(n / weight);
                n -= (digit * weight);
                *(c++) = '0' + digit;
            }
            if (m == 0 && n > 0)
            {
                   *(c++) = ',';
```

```
            }
            m--;
        }
        if (useExp) {
            // convert the exponent
            int i, j;
            *(c++) = 'e';
            if (m1 > 0) {
                *(c++) = '+';
            } else {
                *(c++) = '-';
                m1 = -m1;
            }
            m = 0;
            while (m1 > 0) {
                *(c++) = '0' + m1 % 10;
                m1 /= 10;
                m++;
            }
            c -= m;
            for (i = 0, j = m-1; i<j; i++, j--) {
                // swap without temporary
                c[i] ^= c[j];
                c[j] ^= c[i];
                c[i] ^= c[j];
            }
            c += m;
        }
        *(c) = '\0';
    }
    return s;
}

// Displays on screen the results of the calculations
void display(char type, int rate, SHOOTREADER *shootreader2, TIMES *timesp, double Hmax,
double m, double mmin, char *filename)
{
    double ovs=0;
    double ovsb1=0;
    double ovsb2=0;

    switch (type)
    {
        case 'A':
        {
            fprintf(stdout,"---RESULTS------------\n");
            fprintf(stdout,"Type A - Bitrate");
            if(rate==106) fprintf(stdout,"fc/128 (%d kbps) \n",rate);
            else if(rate==212) fprintf(stdout,"fc/64 (%d kbps) \n",rate);
            else if(rate==424) fprintf(stdout,"fc/32 (%d kbps) \n",rate);
            else if(rate==848) fprintf(stdout,"fc/16 (%d kbps) \n",rate);
            ovs=(((shootreader2->above)-Hmax)/(Hmax-timesp->a))*100+100;
            if (ovs<0)    ovs=0;

            switch (rate)
            {
                case (106):
                {
                    fprintf(stdout,"---Timings------------\n");
                    fprintf(stdout,"t1 = %.2f/fc (%.2f ns)\n", (timesp->t1)*13.56e6,
(timesp->t1)*1e9);
                    fprintf(stdout,"t2 = %.2f/fc (%.2f ns)\n", (timesp->t2)*13.56e6,
(timesp->t2)*1e9);
                    fprintf(stdout,"t3 = %.2f/fc (%.2f ns)\n", (timesp->t3)*13.56e6,
(timesp->t3)*1e9);
                    fprintf(stdout,"t4 = %.2f/fc (%.2f ns)\n", (timesp->t4)*13.56e6,
(timesp->t4)*1e9);

                    fprintf(stdout,"---Overshoot----------\n");
                    fprintf(stdout,"Overshoot = %.2f %% of Hinitial\n", ovs);
```

```
                    fprintf(stdout,"---Envelope-----------\n");
                    fprintf(stdout,"Initial Value = %.5f V(peak-to-peak) \n", Hmax*2);
                    fprintf(stdout,"During 60%% of t2 (max/avg/min): %.5f %% / %.5f %% /
%.5f %% \n",(timesp->a_max)/Hmax*100,(timesp->a)/Hmax*100,(timesp->a_min)/Hmax*100);

                    fprintf(stdout,"---Monotonicity-------\n");
                    if(timesp->mono==0)                fprintf(stdout,"No Monotony Issues
\n");
                    else if (timesp->mono==-1)        fprintf(stdout,"Hump exceeds 5%%
during t2 \n");
                    else if (timesp->mono>0)         fprintf(stdout,"Time between local
max and previous point with the same value = %.2f (ns)\n",(timesp->mono)*1e9);
                    else if (timesp->mono<0)
                    {
                        timesp->mono=timesp->mono*-1;
                        fprintf(stdout,"Hump exceeds 5%% during t2 and time between local
max and previous point with the same value = %.2f (ns)\n",(timesp->mono)*1e9);
                    }
                }
                break;
                case (212):
                case (424):
                case (848):
                {
                    fprintf(stdout,"---Timings------------\n");
                    fprintf(stdout,"t1 = %.2f/fc (%.2f ns)\n", (timesp->t1)*13.56e6,
(timesp->t1)*1e9);
                    fprintf(stdout,"t5 = %.2f/fc (%.2f ns)\n", (timesp->t5)*13.56e6,
(timesp->t5)*1e9);
                    fprintf(stdout,"t6 = %.2f/fc (%.2f ns)\n", (timesp->t6)*13.56e6,
(timesp->t6)*1e9);

                    fprintf(stdout,"---Overshoot----------\n");
                    fprintf(stdout,"hovs = %.3f \n", (((shootreader2->above-Hmax)/Hmax)));
// OVS in (no unit)

                    fprintf(stdout,"---Envelope-----------\n");
                    fprintf(stdout,"Initial Value = %.5f V(peak-to-peak) \n", Hmax*2);
                    fprintf(stdout,"a = %f \n", ((timesp->a)/Hmax));

                    fprintf(stdout,"---Monotonicity-------\n");
                    if(timesp->mono!=0)            fprintf(stdout,"Max. Difference between
local max and previous min = %.3f \n",timesp->mono);
                    else if (timesp->mono==0)     fprintf(stdout,"No Monotony Issues \n");
                }
                break;
            }
        }
        break;

        case 'B':
        {
            ovsb1=(timesp->b-(shootreader2->above_b))*1000;
            if (ovsb1<0)    ovsb1=0;
            ovsb2=((shootreader2->above)-Hmax)*1000;
            if (ovsb2<0)    ovsb2=0;

            if (rate>848)
                type='V';

            fprintf(stdout,"---RESULTS------------\n");
            if(rate==106) fprintf(stdout,"Type B - Bitrate fc/128 (%d kbps)\n", rate);
            else if(rate==212) fprintf(stdout,"Type B - Bitrate fc/64 (%d kbps)\n", rate);
            else if(rate==424) fprintf(stdout,"Type B - Bitrate fc/32 (%d kbps)\n", rate);
            else if(rate==848) fprintf(stdout,"Type B - Bitrate fc/16 (%d kbps)\n", rate);
            else if(rate==1700) fprintf(stdout,"Type VHBR - Bitrate fc/8 (%d kbps)\n",
rate);
            else if(rate==3400) fprintf(stdout,"Type VHBR - Bitrate fc/4 (%d kbps)\n",
rate);
            else if(rate==6800) fprintf(stdout,"Type VHBR - Bitrate fc/2 (%d kbps)\n",
rate);
```

```
            fprintf(stdout,"---Timings------------\n");
            fprintf(stdout,"tf = %.2f/fc (%.2f ns)\n", (timesp->tf)*13.56e6, (timesp-
>tf)*1e9);
            fprintf(stdout,"tr = %.2f/fc (%.2f ns)\n", (timesp->tr)*13.56e6, (timesp-
>tr)*1e9);
            if (rate>848)
            {
                fprintf(stdout,"tr_max = %.2f/fc (%.2f ns)\n", (timesp->tfmax)*13.56e6,
(timesp->tfmax)*1e9);
                fprintf(stdout,"tr_min = %.2f/fc (%.2f ns)\n", (timesp->tfmin)*13.56e6,
(timesp->tfmin)*1e9);
                fprintf(stdout,"tr_N = %.1f  \n", timesp->tf_counter);
                fprintf(stdout,"tf_max = %.2f/fc (%.2f ns)\n", (timesp->trmax)*13.56e6,
(timesp->trmax)*1e9);
                fprintf(stdout,"tf_min = %.2f/fc (%.2f ns)\n", (timesp->trmin)*13.56e6,
(timesp->trmin)*1e9);
                fprintf(stdout,"tf_N = %.1f  \n", timesp->tr_counter);
            }

            fprintf(stdout,"---Overshoots---------\n");
            fprintf(stdout,"hf = %.2f \n", (shootreader2->hf_reader));
            fprintf(stdout,"hr = %.2f \n", (shootreader2->hr_reader));

            fprintf(stdout,"---Envelope----------\n");
            fprintf(stdout,"Initial Value = %.2f V(peak-to-peak) \n", Hmax*2);

            fprintf(stdout,"---Modulation index----\n");
            fprintf(stdout,"m = %.2f %% \n", m);
            if((rate==1700 || rate==3400 || rate==6800))
                fprintf(stdout,"m_min = %.2f %% \n", mmin);

            fprintf(stdout,"---Monotonicity-------\n");
            if(timesp->mono==-1)    fprintf(stdout,"Falling edge not monotonic \n");
            else if(timesp->mono==-2)    fprintf(stdout,"Rising edge not monotonic \n");
            else if(timesp->mono==-3)    fprintf(stdout,"Falling and rising edges not
monotonic \n");
            else                     fprintf(stdout,"No Monotony Issues \n");
        }
        break;
    }
}

// Main Function
int main (int argc, char *argv[])
{
    char type;
    int rate;
    char voltstr[25];
    // intermediate char array to modify the voltage values
    char timestr[25];
    // intermediate char array to modify the time values
    double snum=0;
    double tnum=0;
    double t=0;
    int filterlength=0;
    double Hmax=0;
    double HmaxVHBR=0;
    double Hmin=0;
    double Hmax2=0;
    double Hmin2=0;
    FILE *pointfile=NULL;
    FILE *input_u2=NULL;
    FILE *poutput=NULL;
    double m=0.0;
    double mmin=0.0;
    int length=0;
    double val=0;
    int posval=0;
    int negval=0;
    double tini=0;
```

```
    double tfin=0;
    int samples=0;
    int out_i=0;
    int length_total=0;
    int sample_ini=0;
    int sample_end=0;
    int flag_cut=0;
    int samplesp=0;
    int fi=0;         // Filter generic index
    double b1=0;      // Filter parameters
    double b2=0;
    double b3=0;
    double b4=0;
    double b5=0;
    double a1=0;
    double a2=0;
    double a3=0;
    double a4=0;
    double a5=0;
    double freq1=0;
    double freq2=0;
    double as[5]={0};
    double bs[5]={0};
    double t0=0;
    double tlast=0;
    int lineskip=0;

    double tei=0;
    int size=0;

    if (argc!=4)
    {
        fprintf(stdout, "Usage: \n");
        fprintf(stdout, "      wdt TYPE BITRATE FILE_NAME \n\n");
        fprintf(stdout, "Parameters: \n");
        fprintf(stdout, "      TYPE        A, B or V (for VHBR) \n");
        fprintf(stdout, "      BITRATE        106, 212, 424 or 844 for types A or B \n");
        fprintf(stdout, "                  1700, 3400 or 6800 for type V \n");
        fprintf(stdout, "      FILE_NAME        the file where your waveform is stored
(including path) \n\n");
        fprintf(stdout, "Examples: \n");
        fprintf(stdout, "      > wdt B 106 C:\\reader_XYZ\\analysis\\waveforms\\
scope_48_2.csv \n");
        fprintf(stdout, "      > wdt A 424 G:\\temp\\wvf\\A424_0mm.csv \n");
        fprintf(stdout, "      > wdt V 3400 E:\\vhbr_22.csv \n");
    }
    else
    {
        double *voutput=malloc (sizeof(double)*MAX_SAMPLES);
        double *toutput=malloc (sizeof(double)*MAX_SAMPLES);
        double *envelope=malloc (sizeof(double)*MAX_SAMPLES);
        double *vfilter=malloc (sizeof(double)*MAX_SAMPLES);
        double *tfilter=malloc (sizeof(double)*MAX_SAMPLES);
        TIMES *timesp=(TIMES *)malloc(sizeof(TIMES));
        TIMES *timesp2=(TIMES *)malloc(sizeof(TIMES));
        SHOOTREADER *shootreader2=(SHOOTREADER *)malloc(sizeof(SHOOTREADER));
        // debugger=atoi(argv[4]);
        // Global variable - commented, otherwise crashes
        // when input parameters not correct

        if (voutput!=NULL && toutput!=NULL && envelope!=NULL && vfilter!=NULL &&
tfilter!=NULL && timesp!=NULL && timesp2!=NULL && shootreader2!=NULL)
        {
            memset(voutput, 0, MAX_SAMPLES);
            memset(toutput, 0, MAX_SAMPLES);
            memset(envelope, 0, MAX_SAMPLES);
            memset(vfilter, 0, MAX_SAMPLES);
            memset(tfilter, 0, MAX_SAMPLES);

            type=*argv[1];
            rate=atoi(argv[2]);
```

```
            if (type!='A' && type!='B' && type!='V')
                fprintf(stdout, "Wrong Type, use A, B or V (for VHBR)");
            else if ((type=='A' || type=='B') && (rate!=106 && rate!=212 && rate!=424 &&
rate!=848))
                fprintf(stdout, "Wrong Bitrate, use 106, 212, 424 or 848");
            else if ((type=='V') && (rate!=1700 && rate!=3400 && rate!=6800))
                fprintf(stdout, "Wrong Bitrate, use 1700, 3400 or 6800");
            else
            {
                if (type=='V')
                    type='B';
                pointfile=fopen(argv[3],"r");

                if(debugger)
                    input_u2=fopen("pre_Hilbert.txt","w");
// modified-intermediate amplitude vector
                if((pointfile!=NULL && !debugger)|| (pointfile!=NULL && debugger &&
input_u2!=NULL))
                {
// 1. LOAD DATA + CHECKING DATA (WITHOUT FILTER)
                    for (lineskip=0; lineskip<10; lineskip++)
// Skips the first 10 lines which are the header of csv files
                    {
                        skip_line (pointfile);
                    }
                    read_line(pointfile,voltstr,timestr);
                    t0=atof(timestr);
                    while (!feof(pointfile) && samplesp<MAX_SAMPLES)
// We are reading the lines of the voltage input file
                    {
                        if (voltstr[0]!='\0')
                        {
                            snum=atof(voltstr);
                            tnum=atof(timestr);
                            if(snum<0)
                                negval++;
                            else
                                posval++;
                            vfilter[samplesp]=snum;
                            tfilter[samplesp]=tnum;
                            samplesp++;
                            read_line(pointfile,voltstr,timestr);
                        }
                        tlast=tfilter[samplesp-1];
                    }
                    samplesp=samplesp+3;

                    if (samplesp>=MAX_SAMPLES*1.01)
// Check if too many sample points (<101% of MAX_SAMPLES)
                        fprintf(stdout,"Too many sample points");
                    else
                    {
                        samplesp=datacheck(posval, negval, samplesp, tlast, pointfile);

                        tlast=tfilter[samplesp];

// 2. DATA FILTER 10 MHz BANDPASS (ODER 20 MHz for VHBR)
                        if (rate==106 || rate==212 || rate==424 || rate==848)
                        {
                            freq1=8.56e6/(1/(2*((tlast-t0)/(samplesp-1))));
                            freq2=18.56e6/(1/(2*((tlast-t0)/(samplesp-1))));
                        }
                        else if (rate==1700 || rate==3400 || rate==6800)
                        {
                            freq1=6.06e6/(1/(2*((tlast-t0)/(samplesp-1))));
                            freq2=21.06e6/(1/(2*((tlast-t0)/(samplesp-1))));
                        }

                        butterworth_coeffs(freq1, freq2, as, bs);
                        b1=bs[0];
                        b2=bs[1];
```

```
                        b3=bs[2];
                        b4=bs[3];
                        b5=bs[4];
                        a1=as[0];
                        a2=as[1];
                        a3=as[2];
                        a4=as[3];
                        a5=as[4];

                        for (fi=0; fi<samplesp; fi++)
                        {
                            if (fi<7 || fi>samplesp-7)
                                voutput[fi]=0;
                            else
                                voutput[fi]=(b1*vfilter[fi]+b2*vfilter[fi-
1]+b3*vfilter[fi-2]+
                                        b4*vfilter[fi-3]+b5*vfilter[fi-4]-
a2*voutput[fi-1]-
                                        a3*voutput[fi-2]-a4*voutput[fi-3]-
a5*voutput[fi-4])/a1;
                        }

                        rewind (pointfile);
                        lineskip=0;
                        for (lineskip=0; lineskip<10; lineskip++)
// Skips the first 10 lines (header of csv files)
                        {
                            skip_line (pointfile);
                        }

                        size=samplesp-7;
                        for (fi=0; fi<(samplesp-7); fi++)
// Reading the lines of the voltage input file
                        {
                            val=voutput[fi];
                            read_line (pointfile,voltstr,timestr);
                            if(debugger)
                                fprintf(input_u2,"%s,%f\n",timestr,val);

                            tei=(double)atof(timestr);
                            toutput[fi]=tei;

                            length++;
// length never used?
                        }

// 3. HILBERT TRANSFORM AND THE COMPLEX ENVELOPE
                        rewind(input_u2);
                        hilbert("pre_Hilbert.txt",size,voutput,toutput,debugger);
// performs Hilbert transform
                        if(debugger)
                        {
                            poutput=fopen("output.txt","r");
// Hilbert transform output vector

                            read_line (poutput,voltstr,timestr);
                            tini=atof(timestr);
                            rewind (poutput);

                            if(poutput!=NULL)
                            {
                                while (!feof(poutput))
// We are reading the lines of the voltage input file
                                {
                                    read_line (poutput,voltstr,timestr);
                                    if (timestr[0]!='\0')
                                    {
                                        snum=atof(voltstr);
                                        voutput[samples]=snum;
                                        t=atof(timestr);
                                        toutput[samples]=t;
```

```
                                        samples++;// ==>US
// Same variable as the one in Hmaxfinder
                                        tfin=t;
                                    }
                                }
                            }
                            else
                                fprintf(stdout,"Error in Hilbert transform\n");
                            fclose(poutput);
                        }
                        else
                        {
                            tini=toutput[0];
                            tfin=toutput[size-1];
                            samples = size;
                        }

// 4. USING A SMOOTHING FILTER (MOV. AVG) TO REDUCE THE NOISE
                        if (rate==106 || rate==212 || rate==424 || rate==848)
                            filterlength=1;
// BUG FOUND - CHANGED FROM 3 TO 1 (APRIL 2016)
                        else if (rate==1700 || rate==3400 || rate==6800)
                            filterlength=1;
// BUG FOUND - THIS VALUE IS OVERRIDDEN IN envfilt() (APRIL 2016)
                        length_total=envfilt(rate, voutput, toutput, filterlength, tini,
tfin, samples, envelope);

// 5. 100% OF H_INITIAL
                        Hmaxfinder(envelope, &Hmax, &Hmin, length_total);

// 6. COMPUTING THE ISO BASED TIMES
                        tfinder(type,envelope,toutput,tini,Hmax,Hmin,rate,length_
total,timesp);

// 6.5. M_min FOR VHBR TYPE B
                        if (rate==1700 || rate==3400 || rate==6800)
                            Mminfinder(envelope, Hmax, Hmin, &HmaxVHBR, timesp, length_
total);

// 7. CHECKING FOR ISO DEFINED MONOTONY
                        if (rate==106 || rate==212 || rate==424 || rate==848)
                            monocheck(envelope, toutput, Hmax, timesp, rate, type);

                        out_i=0;
                        while (out_i<MAX_SAMPLES)
// Finds how many zeros are at the beginning of vector envelope
                        {
                            if (envelope[out_i]==0 && flag_cut==0)
                            {
                                sample_ini=out_i;
                                tini=toutput[sample_ini+1];
                            }

                            if (envelope[out_i]!=0)
                            {
                                flag_cut=1;
                                sample_end=out_i;
                                tfin=toutput[sample_end];
                            }
                            out_i++;
                        }

                        samples=sample_end-sample_ini-1; // ==>US

                        for (out_i=0; out_i<samples; out_i++)
                        {
                            voutput[out_i]=envelope[out_i+sample_ini+1];
                            toutput[out_i]=toutput[out_i+sample_ini+1];
                        }
                        for (out_i=samples+1; out_i<MAX_SAMPLES; out_i++)
                        {
```

```
                            voutput[out_i]=0.0;
                            toutput[out_i]=0.0;
                        }

                        tini=toutput[0];
                        tfin=toutput[samples];

// 8. OVERSHOOT OF THE READER
                        fprintf (stdout,"\n");
// 2nd set of functions, "New Line" printed for debug purposes
                        if (rate==106 || rate==212 || rate==424 || rate==848)
                        {
                            filterlength=3;
                            length_total=envfilt(rate, voutput, toutput, filterlength,
tini, tfin, samples, envelope);
// 2nd Filtering to find the alternate envelope
                            Hmaxfinder(envelope, &Hmax2, &Hmin2, length_total);
                            tfinder(type,envelope,toutput,tini,Hmax2,Hmin2,rate,length_
total, timesp2);
                            overshoot(timesp2,Hmax2,envelope,toutput,rate,type,samples,sho
otreader2);    // This time the over- and undershoots are found
                        }
                        else if (rate==1700 || rate==3400 || rate==6800)
                        {
                            filterlength=3;
                            length_total=envfilt(106, voutput, toutput, filterlength,
tini, tfin, samples, envelope);
// 2nd Filtering to find the alternate envelope
                            Hmaxfinder(envelope, &Hmax2, &Hmin2, length_total);
                            overshoot(timesp,Hmax,envelope,toutput,rate,type,samples,shoot
reader2);    // This time the over- and undershoots are found
                        }

// 9. MODULATION
                        m=modulation(type, Hmax, timesp->b);
                        if((type=='B') && (rate==1700 || rate==3400 || rate==6800))
                            mmin=modulation(type,HmaxVHBR, timesp->bVHBR);

// 10. DISPLAY
                        display(type, rate, shootreader2, timesp, Hmax, m, mmin, argv[3]);
                    }
                }
                else if (pointfile==NULL || input_u2!=NULL)
                    fprintf(stdout,"file(s) could not be opened \n");

                fclose(pointfile);
                fclose(input_u2);
            }
        }

        else
            fprintf(stdout, "Memory could not be allocated");

        free (voutput);
        free (toutput);
        free (envelope);
        free (vfilter);
        free (tfilter);
        free (timesp);
        free (timesp2);
        free (shootreader2);

    }

    return 0;
}
```

Annex F
(informative)

Program for the evaluation of the load modulation amplitude

The following program written in C language gives an example for the calculation of the amplitude of the load modulation.

```
/*************************************************************/
/***  This program calculates the Fourier coefficients       ***/
/***  of load modulated voltage of a PICC according          ***/
/***  the ISO/IEC 10373-6 Test methods                       ***/
/***  The coefficients are calculated at the frequencies:    ***/
/***    Carrier:        Fcm (=13.5600 for 13.56 MHz)         ***/
/***    Upper sideband: Fcm + fs                             ***/
/***    Lower sideband: Fcm - fs                             ***/
/***  fs is the subcarrier frequency and its value is:       ***/
/***  Fcm/16 for bit rates up to fc/16, Fcm/8 for a bit rate ***/
/*** of fc/8, Fcm/4 for a bit rate of fc/4 or Fcm/2 for a    ***/
/*** bit rate of fc/2                                       ***/
/*************************************************************/
/*** Input:                                                 ***/
/***  File in CSV Format containing a Table of two          ***/
/***  columns (time and test PCD output voltage vd, clause 7)***/
/***                                                        ***/
/***  data format of input-file:                            ***/
/***  -------------------------                             ***/
/***  - one data-point per line:                            ***/
/***    (time[seconds], sense-coil-voltage[volts])          ***/
/***  - contents in ASCII, no headers                       ***/
/***  - data-points shall be equidistant in time            ***/
/***  - modulation waveform centered                        ***/
/***    (max. tolerance: half of subcarrier cycle)          ***/
/***                                                        ***/
/***                                                        ***/
/***  example for spreadsheet file (start in next line):    ***/
/***   (time)      (voltage)                                ***/
/***  3.00000e-06,1.00                                      ***/
/***  3.00200e-06,1.01                                      ***/
/***  .....                                                 ***/
/*************************************************************/
/*** RUN:                                                   ***/
/***  "exefilename" [filename1[.csv] SubcarrierCode ]       ***/
/*************************************************************/
/*** ISO/IEC 10373-6 DFT CALCULATION                        ***/
/*** Version history:                                       ***/
/*** JUL 2000, version 1.1: original published version      ***/
/*** APR 2008, version 2.0: add the Bartlett window         ***/
/*** NOV 2008, version 2.1: published version with revision ***/
/*** SEP 2010, version 3.0: support higher subcarrier freq. ***/
/*************************************************************/

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

#define MAX_SAMPLES 50000
#define MAX_POINTS 500
#define MAX_MOYENNE 200

double  pi;   /* pi=3.14.... */

/* Array for time and sense coil voltage vd */
double vtime[MAX_SAMPLES]; /* time array                      */
```

```
double vd[MAX_SAMPLES];    /* Array for different coil voltage */


/*************************************************************/
/***    Read CSV File  Function                            ***/
/***                                                       ***/
/***    Description:                                       ***/
/***    This function reads the Table of time and sense coil ***/
/***    voltage from a File in CSV Format                  ***/
/***                                                       ***/
/***    Input: filename                                    ***/
/***                                                       ***/
/***    Return: Number of samples  (sample Count)          ***/
/***            0  if an error occurred                    ***/
/***                                                       ***/
/***    Displays Statistics:                               ***/
/***                                                       ***/
/***    Filename, SampleCount, Sample rate, Max/Min Voltage  ***/
/*************************************************************/

int readcsv(char* fname)
{
   double a,b;
   double max_vd,min_vd;
   int i;
   FILE     *sample_file;

   /************ Open File **********************************/
   if (!strchr(fname, '.'))   strcat(fname, ".csv");

   if ((sample_file = fopen(fname, "r"))== NULL)
      {
             printf("Cannot open input file %s.\n",fname);
             return 0;
      }
   /*********************************************************/
   /* Read CSV File                                         */
   /*********************************************************/
   max_vd=-1e-9F;
   min_vd=-max_vd;
   i=0;

   while (!feof(sample_file))
      {
      if (i>=MAX_SAMPLES)
        {
        printf("Warning: File truncated !!!\n");
        printf("To much samples in file %s\b\n",fname);
        break;
        }
/*      fscanf(sample_file,"%Lf,%Lf\n", &a, &b); */
/* correction of type assignment (no change in algo) WG8 N3230 */
        fscanf(sample_file,"%lf,%lf\n", &a, &b);
      vtime[i] = a;
      vd[i] = b;
      if (vd[i]>max_vd) max_vd=vd[i];
      if (vd[i]<min_vd) min_vd=vd[i];
      i++;
      }
   fclose(sample_file);

   /************ Displays Statistics  ***********************/
   printf("\n*************************************************\n");

   printf("\nStatistics: \n");
   printf(" Filename    : %s\n",fname);
   printf(" Sample count: %d\n",i);
   printf(" Sample rate : %1.0f MHz\n",1e-6/(vtime[1]-vtime[0]));
   printf(" Max(vd)     : %4.0f mV\n",max_vd*1000);
   printf(" Min(vd)     : %4.0f mV\n",min_vd*1000);
```

```
   return i;
}/*************** End ReadCsv **************/


/*************************************************************/
/***    DFT : Discrete Fourier Transformation              ***/
/*************************************************************/
/***    Description:                                       ***/
/***    This function calculate the Fourier coefficient    ***/
/***                                                       ***/
/***    Input: Number of samples                           ***/
/***           Carrier divider of the subcarrier           ***/
/***                                                       ***/
/***    Global Variables:                                  ***/
/***                                                       ***/
/***    Displays Results:                                  ***/
/***                                                       ***/
/***     Carrier coefficient                               ***/
/***     Upper sideband coefficient                        ***/
/***     Lower sideband coefficient                        ***/
/***                                                       ***/
/*************************************************************/
void dft(int count, int CarrierDivider)
{
  double c0_real,c0_imag,c0_abs,c0_phase;
  double c1_real,c1_imag,c1_abs,c1_phase;
  double c2_real,c2_imag,c2_abs,c2_phase;
  int N_data,center,start;
  double w0,wu,wl;
  double Wb;        /* Bartlett window coefficient */

  int i,k;

  double fc;        /* add variable for carrier frequency */


  fc=13.56e6;

  w0=(double)(fc*2.0)*pi; /* carrier 13.56 MHz */
  wu=(double)(1.0+1.0/CarrierDivider)*w0; /* upper sideband 14.41 MHz */
  wl=(double)(1.0-1.0/CarrierDivider)*w0; /* lower sideband 12.71 MHz */
  c0_real=0; /* real part of the carrier fourier coefficient */
  c0_imag=0; /* imag part of the carrier fourier coefficient */
  c1_real=0; /* real part of the up. sideband fourier coefficient */
  c1_imag=0; /* imag part of the up. sideband fourier coefficient */
  c2_real=0; /* real part of the lo. sideband fourier coefficient */
  c2_imag=0; /* imag part of the lo. sideband fourier coefficient */

  center=(count+1)/2; /* center address */

  /********** signal selection ****************************/

  /* Number of samples for six subcarrier periods */

  N_data=(int)(0.5+6.0F*CarrierDivider/(vtime[2]-vtime[1])/fc);

  /* Note: (vtime[2]-vtime[1]) is the scope sample rate */

  start=center - (int) N_data / 2;

  /****************** DFT ********************************/

  for( i=0;i<=N_data-1;i++)
  {
  /* Bartlett window */
  if ((N_data & 1) == 0)
  {
    /* N_data is even */
    if (i < (int) N_data /2)
    {
      Wb=2.0F*i/(double)(N_data - 1);
```

```
    }
    else
    {
      Wb=2.0F*(N_data-i-1)/(double)(N_data - 1);
    }
  }
  else
  {
    /*N_data is odd */
    if (i < (int) N_data /2)
    {
      Wb=2.0F*i/(double)(N_data - 1);
    }
    else
    {
      Wb=2.0F-2.0F*i/(double)(N_data - 1);
    }

  }

  k=i+start;

  c0_real=c0_real+vd[k]*(double)cos(w0*vtime[k])*Wb;
  c0_imag=c0_imag+vd[k]*(double)sin(w0*vtime[k])*Wb;
  c1_real=c1_real+vd[k]*(double)cos(wu*vtime[k])*Wb;
  c1_imag=c1_imag+vd[k]*(double)sin(wu*vtime[k])*Wb;
  c2_real=c2_real+vd[k]*(double)cos(wl*vtime[k])*Wb;
  c2_imag=c2_imag+vd[k]*(double)sin(wl*vtime[k])*Wb;
  }

  /****************** DFT scale *************************/

  c0_real=4.0F*c0_real/(double) N_data;
  c0_imag=4.0F*c0_imag/(double) N_data;
  c1_real=4.0F*c1_real/(double) N_data;
  c1_imag=4.0F*c1_imag/(double) N_data;
  c2_real=4.0F*c2_real/(double) N_data;
  c2_imag=4.0F*c2_imag/(double) N_data;
  /* Note: 4.0F includes the correction coef. of the bartlett window */

  /**************** absolute fourier coefficient **********/
  c0_abs=(double)sqrt(c0_real*c0_real + c0_imag*c0_imag);
  c1_abs=(double)sqrt(c1_real*c1_real + c1_imag*c1_imag);
  c2_abs=(double)sqrt(c2_real*c2_real+c2_imag*c2_imag);

  /************** Phase of fourier coefficient **********/
  c0_phase=(double)atan2(c0_imag,c0_real);
  c1_phase=(double)atan2(c1_imag,c1_real);
  c2_phase=(double)atan2(c2_imag,c2_real);

  /************** Result Display *************************/

  printf("\n\nResults: \n");
  printf("Subcarrier frequency = fc/%d\n",CarrierDivider);
  printf("Carrier ");
  printf("Abs: %7.3fmV ",1000*c0_abs);
  printf("Phase: %3.0fdeg\n",c0_phase/pi*180);
  printf("Upper sideband ");
  printf("Abs: %7.3fmV ",1000*c1_abs);
  printf("Phase: %3.0fdeg\n",c1_phase/pi*180);
  printf("Lower sideband ");
  printf("Abs: %7.3fmV ",1000*c2_abs);
  printf("Phase: %3.0fdeg\n\n",c2_phase/pi*180);
  printf("\n***********************************************\n");
  return;
  }/*************** End DFT **************/


/*************************************************************/
/***    MAIN  Program                                      ***/
/*************************************************************/
```

```
int main(unsigned short paramCount,char *paramList[])
{
   char fname[256];
   unsigned int  sample_count;
   int Dsi; /* Subcarrier frequency code */
   int Ds[4]={2, 4, 8, 16}; /* Carrier divider */

   pi = (double)atan(1.0)*4; /* calculate pi */


   printf("\n***********************************************\n");
   printf("\n**** ISO/IEC 10373-6 PICC Test-Program     ****\n");
   printf("\n**** Version: 3.0 SEPTEMBER 2010           ****\n");
   printf("\n****                      ****\n");
   printf("\n***********************************************\n");
   /************* No Input Parameter           *****************/
   if (paramCount==1)
   {
      printf("\nCSV File name :");
      scanf("%s",fname);
    printf("\nSubcarrier frequency code [1=fc/2, 2=fc/4, 3=fc/8 or 4=fc/16)] :");
      scanf("%d",&Dsi);
   }
   else
   {
      /************* Input Parameter Loop     *****************/
      strcpy(fname,paramList[1]);
    if (!strchr(fname, '.'))   strcat(fname, ".csv");
    if (paramCount>2)
    {
      Dsi=atoi(paramList[2]);
    }
    else
    {
      Dsi=4; /*** for backward compatibility ***/
    }
   }
   if (!strchr(fname, '.'))   strcat(fname, ".csv");
   if (!(sample_count=readcsv(fname))) return 0;
   if ((Dsi > 0) & (Dsi < 5))
   {
     dft(sample_count,Ds[Dsi-1]);
   }
   else
   {
     printf("\nError: Subcarrier frequency code shall be 1,2,3,or 4\n");
   }

   return 0;
}/********************** End Main  ***************************/
```

Annex G
(normative)

Additional PICC test methods

G.1 PICC-test-apparatus and accessories

G.1.1 General

This clause defines the test apparatus and test circuits for verifying the operation of a PICC according to ISO/IEC 14443-3. The test apparatus includes the following:

1) calibration coil (see 5.3);

2) Test PCD assembly (see 5.4);

3) digital sampling oscilloscope (see 5.2).

Care shall be taken to ensure that the results are not affected by the RF performance of the test circuits.

G.1.2 Emulating the I/O protocol

The PICC-test-apparatus shall be able to emulate the Type A and Type B protocols, which are required to test a PICC.

G.1.3 Generating the I/O character timing in reception mode

The PICC-test-apparatus shall be able to generate the I/O bit stream according to ISO/IEC 14443-3. Timing parameters: start bit length, guard time, bit width, request guard time, start of frame width, end of frame width shall be configurable.

G.1.4 Measuring and monitoring the RF I/O protocol

The PICC-test-apparatus shall be able to measure and monitor the timing of the logical low and high states of the RF Input/Receive line relative to the clock frequency. The PICC-test-apparatus shall be able to monitor the PICC subcarrier.

G.1.5 Protocol Analysis

The PICC-test-apparatus shall be able to analyze the I/O-bit stream in accordance with protocol Type A and Type B as specified in ISO/IEC 14443-3 and ISO/IEC 14443-4 and extract the logical data flow for further protocol analysis.

G.1.6 RFU fields and values

RFU fields should be constantly monitored during the testing and shall always be verified to contain the assigned default value. A test shall be FAIL and the tested PICC shall be declared non-compliant in case an RFU field is not set to its default value at any time.

Functional fields should be constantly monitored during the testing and shall always be verified to contain only functional values documented in the standard or proprietary values documented in the standard. A test shall be FAIL and the tested PICC shall be declared non-compliant in case a functional field is not set to said values (and thus is set to an RFU or restricted value) at any time.

G.1.7 Measuring timing

G.1.7.1 Timing measurements

The PICC-test-apparatus shall continuously monitor the following frame format and timing values:

For PICC Type A:

1) Frame Delay Time PCD to PICC (see ISO/IEC 14443-3:2018, 6.2.1.1), with respect to the minimum FDT and the maximum FDT (= FWT), where the use of the determination method defined in Annex P is not mandatory;

2) frame formats (see ISO/IEC 14443-3:2018, 6.2.3);

3) frame waiting time (see ISO/IEC 14443-4:2018, 7.2).

For PICC Type B:

1) character, frame format and timing (see ISO/IEC 14443-3:2018, 7.1);

2) frame waiting time (see ISO/IEC 14443-4:2018, 7.2).

A test shall be FAIL and the tested PICC shall be declared non-compliant in case one of the listed timing constraints is violated.

G.1.7.2 Timing measurement report

Fill out Table G.110 for PICC Type A and/or Table G.111 for PICC Type B with the measured timing values.

G.2 General considerations

G.2.1 Use of test commands (possible additional PICC responses)

G.2.1.1 General

The test commands are defined in 3.2.

The PICC-test-apparatus shall take into account the following rules.

G.2.1.2 Rules on TEST_COMMAND1 and TEST_RESPONSE1

If TEST_RESPONSE1 is chained, the PICC-test-apparatus adapts by sending corresponding R(ACK) to continue the scenario as specified.

If the PICC sends one or several S(WTX) requests before TEST_RESPONSE1, the PICC-test-apparatus adapts by sending necessary S(WTX) responses to continue the scenario as specified.

G.2.1.3 Rules on TEST_COMMAND2 and TEST_RESPONSE2

If the PICC does not support any command expecting a response consisting of n chained I-blocks, the scenarios using TEST_COMMAND2 are not applicable.

If the PICC sends one or several S(WTX) requests before TEST_RESPONSE2, the PICC-test-apparatus adapts by sending necessary S(WTX) responses to continue the scenario as specified.

G.2.1.4 Rules on TEST_COMMAND3 and TEST_RESPONSE3

If the PICC does not support any command needing more than FWT time for execution, the scenarios using TEST_COMMAND3 are not applicable.

If the PICC sends several S(WTX) requests before TEST_RESPONSE3, the PICC-test-apparatus adapts by sending necessary S(WTX) responses to continue the scenario as specified.

If TEST_RESPONSE3 is chained, the PICC-test-apparatus adapts by sending necessary R(ACK) to continue the scenario as specified.

G.2.2 Relationship of test methods versus base standard requirement

Table G.1 lists the applicable tests for Type A PICCs.

Table G.2 lists the applicable tests for Type B PICCs.

Table G.3 lists the applicable tests for both Type A and Type B PICCs.

The ISO/IEC 14443-4 PICC should also comply with ISO/IEC 14443-3 and should be subjected to both the ISO/IEC 14443-3 and ISO/IEC 14443-4 tests for the applicable communication signal interface.

A PICC compliant with ISO/IEC 14443-3 but not with ISO/IEC 14443-4 and in ACTIVE or ACTIVE* state (see G.3.3.7, G.3.3.12 and G.4.4.7) may respond with any frame (including Mute) to frames not related to ISO/IEC 14443-3.

Table G.1 — Test methods for logical operation of the PICC Type A protocol

Test method from ISO/IEC 10373-6		Corresponding requirement	
Clause	**Name**	**Base standard**	**Clause(s)**
G.3.2	Polling	ISO/IEC 14443-3:2018	5.3
G.3.3	Testing of the PICC Type A state transitions	ISO/IEC 14443-3:2018	6.3, 6.4, 6.5
G.3.4	Handling of Type A anticollision	ISO/IEC 14443-3:2018	6.4.2
G.3.5	Handling of RATS	ISO/IEC 14443-4:2018	5.6.1.2
G.3.6	Handling of PPS request	ISO/IEC 14443-4:2018	5.6.2.2
G.3.7	Handling of FSD	ISO/IEC 14443-4:2018	5.1
G.3.8	Handling of Frame Delay Time PICC to PCD and SFGT	ISO/IEC 14443-3:2018 ISO/IEC 14443-4:2018	6.2.1 5.2.5
G.3.9	PICC bit rates capability	ISO/IEC 14443-3:2018	6.1, 6.2

Table G.2 — Test methods for logical operation of the PICC Type B protocol

Test method from ISO/IEC 10373-6		Corresponding requirement	
Clause	**Name**	**Base standard**	**Clause(s)**
G.4.2	Polling	ISO/IEC 14443-3:2018	5.3
G.4.3	PICC framing and bit rates capability	ISO/IEC 14443-3:2018	7.1
G.4.4	Testing of the PICC Type B state transitions	ISO/IEC 14443-3:2018	7.4 to 7.12
G.4.5	Handling of Type B anticollision	ISO/IEC 14443-3:2018	7.4 to 7.12
G.4.6	Handling of ATTRIB	ISO/IEC 14443-3:2018	7.10
G.4.7	Handling of Maximum Frame Size	ISO/IEC 14443-3:2018	7.10.4
G.4.8	Handling of TR2 and SFGT	ISO/IEC 14443-3:2018 ISO/IEC 14443-3:2018	7.9.4.4 7.9.4.7

Table G.3 — Test methods for logical operation of PICC Type A or Type B

Test method from ISO/IEC 10373-6		Corresponding requirement	
Clause	**Name**	**Base standard**	**Clause(s)**
G.5.2	PICC reaction to ISO/IEC 14443-4 scenarios	ISO/IEC 14443-4:2018	Clause 7

Table G.3 *(continued)*

Test method from ISO/IEC 10373-6		Corresponding requirement	
Clause	**Name**	**Base standard**	**Clause(s)**
G.5.3	Handling of PICC error detection	ISO/IEC 14443-4:2018	7.5.7
G.5.4	PICC reaction on CID	ISO/IEC 14443-4:2018	7.1.2.2
G.5.5	PICC reaction on NAD	ISO/IEC 14443-4:2018	7.1.2.3
G.5.6	PICC reaction on S(PARAMETERS) blocks	ISO/IEC 14443-4:2018	7.5.1

G.3 Test method for initialization of the PICC Type A

G.3.1 General

The tests in this subclause determine whether a PICC Type A conforms to ISO/IEC 14443-3 and the protocol activation sequence in ISO/IEC 14443-4:2018, Clause 5. If compliance with ISO/IEC 14443-4 is not required, all tests containing ISO/IEC 14443-4 commands need not be applied.

G.3.2 Scenario G.1: Polling

G.3.2.1 Scope

This test is to determine the behavior of the PICC Type A on receiving REQA commands according to ISO/IEC 14443-3:2018, 5.3.

G.3.2.2 Procedure

Perform the following steps for 3 different operating field strengths of H_{min}, $(H_{min} + H_{max})/2$ and H_{max} as specified for the respective PICC class:

a) Place the PICC into the field and adjust it.

b) Switch the RF operating field off for a minimum time for resetting a PICC (see ISO/IEC 14443-3:2018, 5.6).

c) Switch the RF operating field on.

d) Do delay of 5 ms and send a valid REQA command frame.

e) Record the presence and the content of the PICC response.

f) Switch the RF operating field off for a minimum time for resetting a PICC (see ISO/IEC 14443-3:2018, 5.6).

g) Switch the RF operating field on.

h) Wait 5 ms and send a valid REQB command frame (using Type B modulation and bit coding).

i) Wait 5 ms and send a valid REQA command frame.

j) Record the presence and the content of the PICC response.

G.3.2.3 Test report

Fill the appropriate row in Table G.112 according to Table G.4.

Table G.4 — Result criteria for Scenario G.1: Polling

Explanation	Test result
Only when the PICC's response is a valid ATQA in both steps e) and j)	PASS
If the PICC's response isn't a valid ATQA in any of steps e) or j)	FAIL

G.3.3 Testing of the PICC Type A state transitions

G.3.3.1 Scope

These tests verify the correct implementation of the PICC Type A state diagram as described in ISO/IEC 14443-3:2018, 6.3.

G.3.3.2 General test outline

G.3.3.2.1 General

For an exhaustive test of the PICC Type A state machine, the correctness of every possible state transition at every state shall be verified. Verifying a specific state using a specific state transition will be done as follows.

First, reset the PICC and place it in the Test Initial State (TIS). This is one of the states from StateSet where the transitions (T) have to be verified. Then execute a transition (T) from TransitionSet. After execution of the state transition, check if the PICC is in the expected Test Target State (TTS). There is a difficulty in how to perform this check, because it is impossible to directly inspect the state machine of the PICC. The solution to this problem is to make some additional state transitions and checking the answer of the PICC. The transitions for this purpose are selected in such way that the state can be determined from the PICC answers as precisely as possible.

G.3.3.2.2 Functions for putting the PICC in the Test Initial State (TIS)

Putting the PICC into the Test Initial State (TIS) will be done by a sequence of transition commands specified in the following tables. The general method is as follows:

In order to put the PICC into the Test Initial State (TIS), lookup the corresponding state transition sequence in Table G.5. Then successively apply the state transitions described in the State Transition Sequence column by looking up the corresponding commands in Table G.6. Always check the content and integrity of the PICC response.

Table G.5 — State transition sequence table

TIS	State transition sequence
POWER-OFF	—
IDLE[a]	POWER-OFF → IDLE
READY(1)	POWER-OFF → IDLE → READY(1)
READY(2)	POWER-OFF → IDLE → READY(1) → READY(2)
READY(3)	POWER-OFF → IDLE → READY(1) → READY(2) → READY(3)
ACTIVE	POWER-OFF → IDLE → READY(1) → … → READY(CascadeLevels) → ACTIVE
PROTOCOL	POWER-OFF → IDLE → READY(1) → … → READY(CascadeLevels) → ACTIVE → PROTOCOL
HALT	POWER-OFF → IDLE → READY(1) → …→ READY(CascadeLevels) → ACTIVE → HALT
READY*(1)	POWER-OFF → IDLE → READY(1) → … → READY(CascadeLevels) → ACTIVE → HALT → READY*(1)

[a] IDLE state may be reached from ACTIVE state.

Table G.5 *(continued)*

TIS	State transition sequence
READY*(2)	POWER-OFF → IDLE → READY(1) → … → READY(CascadeLevels) → ACTIVE → HALT → READY*(1) → READY*(2)
READY*(3)	POWER-OFF → IDLE → READY(1) → … → READY(CascadeLevels) → ACTIVE → HALT → READY*(1) → READY*(2) → READY*(3)
ACTIVE*	POWER-OFF → IDLE → READY(1) → …→ READY (CascadeLevels) → ACTIVE → HALT → READY*(1) → … → READY*(CascadeLevels) → ACTIVE*
[a] IDLE state may be reached from ACTIVE state.	

Table G.6 — State transition table

State → Next state	PICC-test-apparatus		PICC
POWER-OFF → IDLE	Power On (RF Field on)	→	
		←	Mute
IDLE → READY(1)	REQA	→	
		←	ATQA
READY(1) → READY(2)	SELECT(1)[a]	→	
		←	SAK (cascade)
READY(2) → READY(3)	SELECT(2)[a]	→	
		←	SAK (cascade)
READY(CascadeLevels) → ACTIVE	SELECT (CascadeLevels)[a]	→	
		←	SAK (complete)
ACTIVE → PROTOCOL	RATS(0,0)	→	
		←	ATS
ACTIVE → HALT	HLTA	→	
		←	Mute
HALT → READY*(1)	WUPA	→	
		←	ATQA
READY*(1) → READY*(2)	SELECT(1)	→	
		←	SAK (cascade)
READY*(2) → READY*(3)	SELECT(2)	→	
		←	SAK(cascade)
READY*(CascadeLevels) →ACTIVE*	SELECT (CascadeLevels)	→	
		←	SAK (complete)
ACTIVE → IDLE	REQA	→	
		←	Mute
[a] If the PICC UID is unknown, SELECT command may be preceded with an anticollision command to retrieve the PICC UID.			

G.3.3.2.3 Functions for checking the validity of the Test Target State (TTS)

Table G.7 describes the state transitions, which are used to check whether the PICC is in the state S. The content of the PICC answer (i.e. ATQA, SAK …) should be thoroughly checked for ISO/IEC 14443-3 and ISO/IEC 14443-4 conformance. Note, that these tests may cause the PICC to change its state.

The READY(I)/READY*(I) states and the ACTIVE/ACTIVE* states cannot be distinguished with one test run. In order to distinguish the "*"-states from the non-"*"-states, perform the following steps:

a) Rerun the test a second time, without checking the TTS.

b) Send REQA command. The PICC response shall be Mute.

c) Send REQA command.

d) If the PICC response is Mute then the PICC state was a "*"-state.

e) Else the PICC was a non-"*"-state.

The HALT state cannot be distinguished from READY*(I) state and from ACTIVE* state with one test run. In order to distinguish the HALT state perform the following steps:

a) Rerun the test a second time, without checking the TTS.

b) Send WUPA command. The PICC response shall be ATQA.

Table G.7 — Checking the TTS

State S	PICC-test-apparatus		PICC
IDLE	REQA	→	
		←	ATQA[a]
READY(I), I < CascadeLevels	SELECT(I)	→	
		←	SAK (cascade)
READY(I), I = CascadeLevels	SELECT(I)	→	
		←	SAK (complete)
ACTIVE	RATS (0,0)	→	
		←	ATS
PROTOCOL	$I(0)_0$(TEST_COMMAND1(1))	→	
		←	$I(0)_0$(TEST_RESPONSE1(1))
HALT	REQA	→	
		←	Mute
	WUPA	→	
		←	ATQA[a]
READY*(I), I < CascadeLevels	SELECT (I)	→	
		←	SAK (cascade)
READY*(I), I = CascadeLevels	SELECT (I)	→	
		←	SAK (complete)
ACTIVE*	RATS(0,0)	→	
		←	ATS

[a] If the PICC UID is known, send an anticollision command to retrieve the PICC UID. Check that the PICC UID has not changed.

NOTE The block number can be 0 or 1 dependent on block numbering rules, see ISO/IEC 14443-4:2018, 7.5.4.

G.3.3.3 Scenario G.2: Behavior of the PICC Type A in the IDLE state

G.3.3.3.1 Scope

This test is to determine the behavior of the PICC Type A in the IDLE state according to ISO/IEC 14443-3:2018, 6.3.2.

G.3.3.3.2 Procedure

Perform the following steps for every row of Table G.8:

a) Put the PICC into IDLE state. If the UID is unknown, put the PICC in ACTIVE state to retrieve its UID and then put it into IDLE state using ACTIVE → IDLE state transition. Check that the value '88' of the cascade tag CT is not used for uid0 in single size UID.

b) Perform the state transition by sending the command as indicated in the PICC-test-apparatus column.

c) Check if the PICC response is as indicated in the PICC column.

d) Check if the PICC is in the Test Target State (TTS).

Table G.8 — Transitions from IDLE state

Transition	PICC-test-apparatus		PICC	TTS
REQA	REQA	→		READY(1)
		←	ATQA	
WUPA	WUPA	→		READY(1)
		←	ATQA	
HLTA	HLTA	→		IDLE
		←	Mute	
AC (empty)	('93 20')	→		IDLE
		←	Mute	
AC	('93' NVB $UIDTX_1[[1..n_1]]$)[a]	→		IDLE
		←	Mute	
nAC	('93' NVB ~$UIDTX_1[[1..n_1]]$)[a]	→		IDLE
		←	Mute	
SELECT	SELECT(1)	→		IDLE
		←	Mute	
nSELECT	('93 70' ~$UIDTX_1$[[1..32]] BCC CRC_A)	→		IDLE
		←	Mute	
RATS	RATS(0,0)	→		IDLE
		←	Mute	
PPS	PPS(0,0,0)	→		IDLE
		←	Mute	
ISO/IEC 14443-4 command	$I(0)_0$(TEST_COMMAND1(1))	→		IDLE
		←	Mute	
DESELECT	S(DESELECT)	→		IDLE
		←	Mute	
Error condition	('26')[b]	→		IDLE
		←	Mute	

[a] Let $1 \le n_1 \le 32$.

[b] The value is sent in a standard frame and not in a short frame.

G.3.3.3.3 Test report

Fill the appropriate row in Table G.112 according to Table G.9.

Table G.9 — Result criteria for behavior of the PICC Type A in the IDLE state

Explanation	Test result
Only when the PICC responded as indicated in the procedure	PASS
Any other case	FAIL

G.3.3.4 Scenario G.3: Behavior of the PICC Type A in the READY(1) state

G.3.3.4.1 Scope

This test is to determine the behavior of the PICC Type A in the READY state on cascade level 1 according to ISO/IEC 14443-3:2018, 6.3.3.

G.3.3.4.2 Procedure

Perform the following steps for all PICCs and every row of Table G.10:

a) Put the PICC into READY(1) state.

b) Perform the state transition by sending the command as indicated in the PICC-test-apparatus column.

c) Check if the PICC response is as indicated in the PICC column.

d) Check if the PICC is in the Test Target State (TTS).

Table G.10 — Transitions from READY(1) state

Transition	PICC-test-apparatus		PICC	TTS
REQA	REQA	→		IDLE
		←	Mute	
WUPA	WUPA	→		IDLE
		←	Mute	
HLTA	HLTA	→		IDLE
		←	Mute	
AC (wrong parity bit)[g]	('93 20', wrong parity bit)	→		IDLE
		←	Mute	
SELECT[f] (wrong parity bit)[g]	SELECT(1) with wrong parity bit	→		IDLE
		←	Mute	
AC (empty)	('93 20')	→		READY(1)
		←	$UIDTX_1$[[1..32]] BCC	

[a] Let $1 \le n_1 \le 32$, $UIDTX_1[[n_1]] = 0$. If such a number does not exist, the test may be skipped.

[b] Let $1 \le n_2 \le 32$, $UIDTX_1[[n_2]] = 1$. If such a number does not exist, the test may be skipped.

[c] Cascade bit of SAK shall be (0)b for single size UID PICCs and (1)b for double and triple size UID PICCs.

[d] Single size UID PICC shall be in ACTIVE state; double and triple size UID PICCs shall be in READY state.

[e] Let $1 \le n_3 \le 32$.

[f] If the PICC UID is unknown, it may be retrieved by putting the PICC in ACTIVE state, and then applying ACTIVE → IDLE, IDLE → READY(1) and READY(1) → READY(2) state transitions.

[g] The parity error is simulated on the first transmitted byte of the frame by reversing the parity bit.

Table G.10 *(continued)*

Transition	PICC-test-apparatus		PICC	TTS
AC[f] (split after (0)b)	('93' NVB $UIDTX_1[[1..n_1]]$)[a]	→		READY(1)
		←	if n_1 = 32 then (BCC) else ($UIDTX_1[[n_1+1..32]]$ BCC)[a]	
AC[f] (split after (1)b)	('93' NVB $UIDTX_1[[1..n_2]]$)[b]	→		READY(1)
		←	if n_2 = 32 then (BCC) else ($UIDTX_1[[n_2+1..32]]$ BCC)[b]	
nAC[f] (wrong UID)	('93' NVB ~$UIDTX_1[[1..n_3]]$)[e]	→		IDLE
		←	Mute	
SELECT[f]	SELECT(1)	→		ACTIVE or READY[d]
		←	SAK[c]	
nSELECT[f] (wrong UID)	('93 70' ~$UIDTX_1$ BCC CRC_A)	→		IDLE
		←	Mute	
Error condition	('93 70' $UIDTX_1$ BCC ~CRC_A)	→		IDLE
		←	Mute	
ISO/IEC 14443-4 command	$I(0)_0$(TEST_COMMAND1(1))	→		IDLE
		←	Mute	
DESELECT	S(DESELECT)	→		IDLE
		←	Mute	
RATS	RATS(0,0)	→		IDLE
		←	Mute	
PPS	PPS(0,0,0)	→		IDLE
		←	Mute	

[a] Let $1 \le n_1 \le 32$, $UIDTX_1[[n_1]] = 0$. If such a number does not exist, the test may be skipped.

[b] Let $1 \le n_2 \le 32$, $UIDTX_1[[n_2]] = 1$. If such a number does not exist, the test may be skipped.

[c] Cascade bit of SAK shall be (0)b for single size UID PICCs and (1)b for double and triple size UID PICCs.

[d] Single size UID PICC shall be in ACTIVE state; double and triple size UID PICCs shall be in READY state.

[e] Let $1 \le n_3 \le 32$.

[f] If the PICC UID is unknown, it may be retrieved by putting the PICC in ACTIVE state, and then applying ACTIVE → IDLE, IDLE → READY(1) and READY(1) → READY(2) state transitions.

[g] The parity error is simulated on the first transmitted byte of the frame by reversing the parity bit.

G.3.3.4.3 Test report

Fill the appropriate row in Table G.112 according to Table G.11.

Table G.11 — Result criteria for behavior of the PICC Type A in the READY(1) state

Explanation	Test result
Only when the PICC responded as indicated in the procedure	PASS
Any other case	FAIL

G.3.3.5 Scenario G.4: Behavior of the PICC Type A in the READY(2) state

G.3.3.5.1 Scope

This test is to determine the behavior of the PICC Type A in the READY state on cascade level 2 according to ISO/IEC 14443-3:2018, 6.3.3. This test is only for PICCs with double or triple size UID.

G.3.3.5.2 Procedure

Perform the following steps for all PICCs with double and triple size UID and every row of Table G.12:

a) Put the PICC into READY(2) state.

b) Perform the state transition by sending the command as indicated in the PICC-test-apparatus column.

c) Check if the PICC response is as indicated in the PICC column.

d) Check if the PICC is in the Test Target State (TTS).

Table G.12 — Transitions from READY(2) state

Transition	PICC-test-apparatus		PICC	TTS
REQA	REQA	→		IDLE
		←	Mute	
WUPA	WUPA	→		IDLE
		←	Mute	
HLTA	HLTA	→		IDLE
		←	Mute	
AC (wrong parity bit)[g]	('95 20', wrong parity bit)	→		IDLE
		←	Mute	
SELECT[f] (wrong parity bit)[g]	SELECT(2) with wrong parity bit	→		IDLE
		←	Mute	
AC (empty)	('95 20')	→		READY(2)
		←	$UIDTX_2[[1..32]]$ BCC	
AC[f] (split after (0)b)	('95' NVB $UIDTX_2[[1..n_1]]$)[a]	→		READY(2)
		←	if $n_1 = 32$ then (BCC) else ($UIDTX_2[[n_1+1..32]]$ BCC)[a]	

[a] Let $1 \le n_1 \le 32$, $UIDTX_2[[n_1]] = 0$. If such a number does not exist, the test may be skipped.

[b] Let $1 \le n_2 \le 32$, $UIDTX_2[[n_2]] = 1$. If such a number does not exist, the test may be skipped.

[c] Cascade bit of SAK shall be (0)b for double size UID PICCs and (1)b for triple size UID PICCs.

[d] Double size UID PICCs shall be in ACTIVE state; triple size UID PICCs shall be in READY state.

[e] Let $1 \le n_3 \le 32$.

[f] If the PICC UID is unknown, it may be retrieved by putting the PICC in ACTIVE state, and then applying ACTIVE → IDLE, IDLE → READY(1) and READY(1) → READY(2) state transitions.

[g] The parity error is simulated on the first transmitted byte of the frame by reversing the parity bit.

Table G.12 *(continued)*

Transition	PICC-test-apparatus		PICC	TTS
AC[f] (split after (1)b)	('95' NVB $UIDTX_2[[1..n_2]]$)[b]	→		READY(2)
		←	if n_2 = 32 then (BCC) else ($UIDTX_2[[n_2+1..32]]$ BCC)[b]	
nAC[f] (wrong UID)	('95' NVB ~$UIDTX_2[[1..n_3]]$)[e]	→		IDLE
		←	Mute	
SELECT[f]	SELECT(2)	→		ACTIVE or READY[d]
		←	SAK[c]	
nSELECT[f] (wrong UID)	('95 70' ~$UIDTX_2$ BCC CRC_A)	→		IDLE
		←	Mute	
Error condition	('95 70' $UIDTX_2$ BCC ~CRC_A)	→		IDLE
		←	Mute	
ISO/IEC 14443-4 command	$I(0)_0$(TEST_COMMAND1(1))	→		IDLE
		←	Mute	
DESELECT	S(DESELECT)	→		IDLE
		←	Mute	
RATS	RATS(0,0)	→		IDLE
		←	Mute	
PPS	PPS(0,0,0)	→		IDLE
		←	Mute	

[a] Let $1 \le n_1 \le 32$, $UIDTX_2[[n_1]] = 0$. If such a number does not exist, the test may be skipped.

[b] Let $1 \le n_2 \le 32$, $UIDTX_2[[n_2]] = 1$. If such a number does not exist, the test may be skipped.

[c] Cascade bit of SAK shall be (0)b for double size UID PICCs and (1)b for triple size UID PICCs.

[d] Double size UID PICCs shall be in ACTIVE state; triple size UID PICCs shall be in READY state.

[e] Let $1 \le n_3 \le 32$.

[f] If the PICC UID is unknown, it may be retrieved by putting the PICC in ACTIVE state, and then applying ACTIVE → IDLE, IDLE → READY(1) and READY(1) → READY(2) state transitions.

[g] The parity error is simulated on the first transmitted byte of the frame by reversing the parity bit.

G.3.3.5.3 Test report

Fill the appropriate row in Table G.112 according to Table G.13.

Table G.13 — Result criteria for behavior of the PICC Type A in the READY(2) state

Explanation	Test result
If the PICC has a single size UID	Not applicable (N/A)
When the PICC has a double or triple size UID and only when it responded as indicated in the procedure	PASS
Any other case	FAIL

G.3.3.6 Scenario G.5: Behavior of the PICC Type A in the READY(3) state

G.3.3.6.1 Scope

This test is to determine the behavior of the PICC Type A in the READY state according to ISO/IEC 14443-3:2018, 6.3.3. This test is only for PICCs with triple size UID.

G.3.3.6.2 Procedure

Perform the following steps for all PICCs with triple size UID and every row of Table G.14:

a) Put the PICC into READY(3) state.

b) Perform the state transition by sending the command as indicated in the PICC-test-apparatus column.

c) Check if the PICC response is as indicated in the PICC column.

d) Check if the PICC is in the Test Target State (TTS).

Table G.14 — Transitions from READY(3) state

Transitions	PICC-test-apparatus		PICC	TTS
REQA	REQA	→		IDLE
		←	Mute	
WUPA	WUPA	→		IDLE
		←	Mute	
HLTA	HLTA	→		IDLE
		←	Mute	
AC (wrong parity bit)[e]	('97 20', wrong parity bit)	→		IDLE
		←	Mute	
SELECT[d] (wrong parity bit)[e]	SELECT(3) with wrong parity bit	→		IDLE
		←	Mute	
AC (empty)	('97 20')	→		READY(3)
		←	$UIDTX_3[[1..32]]$ BCC	
AC[d] (split after (0)b)	('97' NVB $UIDTX_3[[1..n_1]]$)[a]	→		READY(3)
		←	if $n_1 = 32$ then (BCC) else ($UIDTX_3[[n_1+1..32]]$ BCC)[a]	
AC[d] (split after (1)b)	('97' NVB $UIDTX_3[[1..n_2]]$)[b]	→		READY(3)
		←	if $n_2 = 32$ then (BCC) else ($UIDTX_3[[n_2+1..32]]$ BCC)[b]	
nAC[d] (wrong UID)	('97' NVB ~$UIDTX_3[[1..n_3]]$)[c]	→		IDLE
		←	Mute	

[a] Let $1 \le n_1 \le 32$, UIDTX3[[n_1]] = 0. If such a number does not exist, the test may be skipped.

[b] Let $1 \le n_2 \le 32$, UIDTX3[[n_2]] = 1. If such a number does not exist, the test may be skipped.

[c] Let $1 \le n_3 \le 32$.

[d] If the PICC UID is unknown, it may be retrieved by putting the PICC in ACTIVE state, and then applying ACTIVE → IDLE, IDLE → READY(1), READY(1) → READY(2) and READY(2) → READY(3) state transitions.

[e] The parity error is simulated on the first transmitted byte of the frame by reversing the parity bit.

Table G.14 *(continued)*

Transitions	PICC-test-apparatus		PICC	TTS
SELECT[d]	SELECT(3)	→		ACTIVE
		←	SAK (complete)	
nSELECT[d] (wrong UID)	('97 70' ~$UIDTX_3$ BCC CRC_A)	→		IDLE
		←	Mute	
Error condition	('97 70' $UIDTX_3$ BCC ~CRC_A)	→		IDLE
		←	Mute	
ISO/IEC 14443-4 command	$I(0)_0$(TEST_COMMAND1(1))	→		IDLE
		←	Mute	
DESELECT	S(DESELECT)	→		IDLE
		←	Mute	
RATS	RATS(0,0)	→		IDLE
		←	Mute	
PPS	PPS(0,0,0)	→		IDLE
		←	Mute	

[a] Let $1 \le n_1 \le 32$, UIDTX3[[n_1]] = 0. If such a number does not exist, the test may be skipped.

[b] Let $1 \le n_2 \le 32$, UIDTX3[[n_2]] = 1. If such a number does not exist, the test may be skipped.

[c] Let $1 \le n_3 \le 32$.

[d] If the PICC UID is unknown, it may be retrieved by putting the PICC in ACTIVE state, and then applying ACTIVE → IDLE, IDLE → READY(1), READY(1) → READY(2) and READY(2) → READY(3) state transitions.

[e] The parity error is simulated on the first transmitted byte of the frame by reversing the parity bit.

G.3.3.6.3 Test report

Fill the appropriate row in Table G.112 according to Table G.15.

Table G.15 — Result criteria for behavior of the PICC Type A in the READY(3) state

Explanation	Test result
If the PICC has a single or double size UID	Not applicable (N/A)
When the PICC has a triple size UID and only when it responded as indicated in the procedure	PASS
Any other case	FAIL

G.3.3.7 Scenario G.6: Behavior of the PICC Type A in the ACTIVE state

G.3.3.7.1 Scope

This test is to determine the behavior of the PICC Type A in the ACTIVE state according to ISO/IEC 14443-3:2018, 6.3.4.

G.3.3.7.2 Procedure

Perform the following steps for every row of Table G.16:

a) Put the PICC into ACTIVE state.

b) Perform the state transition by sending the command as indicated in the PICC-test-apparatus column.

c) Check if the PICC response is as indicated in the PICC column.

d) Check if the PICC is in the Test Target State (TTS).

Table G.16 — Transitions from ACTIVE state

Transition	PICC-test-apparatus		PICC	TTS
REQA	REQA	→		IDLE
		←	Mute	
WUPA	WUPA	→		IDLE
		←	Mute	
AC	('93' NVB $UIDTX_1$[[1..n_1]])[a]	→		IDLE
		←	Mute[b]	
nAC	('93' NVB ~$UIDTX_1$[[1..n_1]])[a]	→		IDLE
		←	Mute[b]	
HLTA	HLTA	→		HALT
		←	Mute	
SELECT	SELECT(1)	→		IDLE
		←	Mute[b]	
nSELECT	('93 70' ~$UIDTX_1$ BCC CRC_A)	→		IDLE
		←	Mute[b]	
RATS	RATS(0,0)	→		PROTOCOL
		←	ATS	
Error condition	('E0 00' ~CRC_A)	→		IDLE
		←	Mute[b]	
ISO/IEC 14443-4 command	$I(0)_0$(TEST_COMMAND1(1))	→		IDLE
		←	Mute[b]	
DESELECT	S(DESELECT)	→		IDLE
		←	Mute[b]	
PPS	PPS(0,0,0)	→		IDLE
		←	Mute[b]	
RATS (wrong parity bit)[c]	RATS(0,0) with wrong parity bit	→		IDLE
		←	Mute[b]	
Type B command	REQB	→		IDLE or ACTIVE[d]
		←	Mute	
AC (empty)	('93 20')	→		IDLE
		←	Mute	

[a] Let $1 \leq n_1 \leq 32$.

[b] Mute or proprietary response.

[c] The parity error is simulated on the first transmitted byte of the frame by reversing the parity bit.

[d] Check first IDLE state as TTS. If TTS is not IDLE state, rerun the test a second time to check ACTIVE state.

G.3.3.7.3 Test report

Fill the appropriate row in Table G.112 according to Table G.17.

Table G.17 — Result criteria for behavior of the PICC Type A in the ACTIVE state

Explanation	Test result
Only when the PICC responded as indicated in the procedure	PASS
Any other case	FAIL

G.3.3.8 Scenario G.7: Behavior of the PICC Type A in the HALT state

G.3.3.8.1 Scope

This test is to determine the behavior of the PICC Type A in the HALT state according to ISO/IEC 14443-3:2018, 6.3.5.

G.3.3.8.2 Procedure

Perform the following steps for every row of Table G.18:

a) Put the PICC into HALT state.

b) Perform the state transition by sending the command as indicated in the PICC-test-apparatus column.

c) Check if the PICC response is as indicated in the PICC column.

d) Check if the PICC is in the Test Target State (TTS).

Table G.18 — Transitions from HALT state

Transition	PICC-test-apparatus		PICC	TTS
REQA	REQA	→		HALT
		←	Mute	
WUPA	WUPA	→		READY*(1)
		←	ATQA	
HLTA	HLTA	→		HALT
		←	Mute	
AC (empty)	('93 20')	→		HALT
		←	Mute	
AC	('93' NVB $UIDTX_1[[1..n_1]]$)[a]	→		HALT
		←	Mute	
nAC	('93' NVB ~$UIDTX_1[[1..n_1]]$)[a]	→		HALT
		←	Mute	
SELECT	SELECT(1)	→		HALT
		←	Mute	
nSELECT	('93 70' ~$UIDTX_1$ BCC CRC_A)	→		HALT
		←	Mute	
RATS	RATS(0,0)	→		HALT
		←	Mute	
Error condition	('52') in the standard frame	→		HALT
		←	Mute	

[a] Let $1 \le n_1 \le 32$.

Table G.18 *(continued)*

Transition	PICC-test-apparatus		PICC	TTS
ISO/IEC 14443-4 command	$I(0)_0$(TEST_COMMAND1(1))	→		HALT
		←	Mute	
DESELECT	S(DESELECT)	→		HALT
		←	Mute	
PPS	PPS(0,0,0)	→		HALT
		←	Mute	
[a] Let $1 \le n_1 \le 32$.				

G.3.3.8.3 Test report

Fill the appropriate row in Table G.112 according to Table G.19.

Table G.19 — Result criteria for behavior of the PICC Type A in the HALT state

Explanation	Test result
Only when the PICC responded as indicated in the procedure	PASS
Any other case	FAIL

G.3.3.9 Scenario G.8: Behavior of the PICC Type A in the READY*(1) state

G.3.3.9.1 Scope

This test is to determine the behavior of the PICC Type A in the READY* state of cascade level 1 according to ISO/IEC 14443-3:2018, 6.3.6.

G.3.3.9.2 Procedure

Perform the following steps for every row of Table G.20:

a) Put the PICC into READY*(1) state.

b) Perform the state transition by sending the command as indicated in the PICC-test-apparatus column.

c) Check if the PICC response is as indicated in the PICC column.

d) Check that the PICC is in the Test Target State (TTS).

Table G.20 — Transitions from READY*(1) state

Transition	PICC-test-apparatus		PICC	TTS
REQA	REQA	→		HALT
		←	Mute	
[a] Let $1 \le n_1 \le 32$, $UIDTX_1[[n_1]] = 0$. If such a number does not exist, the test may be skipped.				
[b] Let $1 \le n_2 \le 32$, $UIDTX_1[[n_2]] = 1$. If such a number does not exist, the test may be skipped.				
[c] Cascade bit of SAK shall be (0)b for single size UID PICCs and (1)b for double and triple size UID PICCs.				
[d] Single size UID PICCs shall be in ACTIVE* state; double and triple size UID PICCs should be in READY* state.				
[e] Let $1 \le n_3 \le 32$.				
[f] The parity error is simulated on the first transmitted byte of the frame by reversing the parity bit.				

Table G.20 *(continued)*

Transition	PICC-test-apparatus		PICC	TTS
WUPA	WUPA	→		HALT
		←	Mute	
HLTA	HLTA	→		HALT
		←	Mute	
AC (wrong parity bit)[f]	('93 20', wrong parity bit)	→		HALT
		←	Mute	
SELECT (wrong parity bit)[f]	SELECT(1) with wrong parity bit	→		HALT
		←	Mute	
AC (empty)	('93 20')	→		READY*(1)
		←	$UIDTX_1$[[1..32]] BCC	
AC (split after (0)b)	('93' NVB $UIDTX_1$[[1..n_1]])[a]	→		READY*(1)
		←	if n_1 = 32 then (BCC) else ($UIDTX_1$[[n_1+1..32]] BCC)[a]	
AC (split after (1)b)	('93' NVB $UIDTX_1$[[1..n_2]])[b]	→		READY*(1)
		←	if n_2 = 32 then (BCC) else ($UIDTX_1$[[n_2+1..32]] BCC)[b]	
nAC (wrong UID)	('93' NVB ~$UIDTX_1$[[1..n_3]])[e]	→		HALT
		←	Mute	
SELECT	SELECT(1)	→		ACTIVE *or READY*[d]
		←	SAK[c]	
nSELECT (wrong UID)	('93 70' ~$UIDTX_1$ BCC CRC_A)	→		HALT
		←	Mute	
Error condition	('93 70' $UIDTX_1$ BCC ~CRC_A)	→		HALT
		←	Mute	
ISO/IEC 14443-4 command	$I(0)_0$(TEST_COMMAND1(1))	→		HALT
		←	Mute	
DESELECT	S(DESELECT)	→		HALT
		←	Mute	
RATS	RATS(0,0)	→		HALT
		←	Mute	
PPS	PPS(0,0,0)	→		HALT
		←	Mute	

[a] Let $1 \le n_1 \le 32$, $UIDTX_1$[[n_1]] = 0. If such a number does not exist, the test may be skipped.

[b] Let $1 \le n_2 \le 32$, $UIDTX_1$[[n_2]] = 1. If such a number does not exist, the test may be skipped.

[c] Cascade bit of SAK shall be (0)b for single size UID PICCs and (1)b for double and triple size UID PICCs.

[d] Single size UID PICCs shall be in ACTIVE* state; double and triple size UID PICCs should be in READY* state.

[e] Let $1 \le n_3 \le 32$.

[f] The parity error is simulated on the first transmitted byte of the frame by reversing the parity bit.

G.3.3.9.3 Test report

Fill the appropriate row in Table G.112 according to Table G.21.

Table G.21 — Result criteria for behavior of the PICC Type A in the READY*(1) state

Explanation	Test result
Only when the PICC responded as indicated in the procedure	PASS
Any other case	FAIL

G.3.3.10 Scenario G.9: Behavior of the PICC Type A in the READY*(2) state

G.3.3.10.1 Scope

This test is to determine the behavior of the PICC Type A in the READY* state of cascade level 2 according to ISO/IEC 14443-3:2018, 6.3.6. This test only applies to PICCs with double or triple size UID.

G.3.3.10.2 Procedure

Perform the following steps for every row of Table G.22:

a) Put the PICC into READY*(2) state.

b) Perform the state transition by sending the command as indicated in the PICC-test-apparatus column.

c) Check if the PICC response is as indicated in the PICC column.

d) Check if the PICC is in the Test Target State (TTS).

Table G.22 — Transitions from READY*(2) state

Transition	PICC-test-apparatus		PICC	TTS
REQA	REQA	→		HALT
		←	Mute	
WUPA	WUPA	→		HALT
		←	Mute	
HLTA	HLTA	→		HALT
		←	Mute	
AC (wrong parity bit)[f]	('95 20', wrong parity bit)	→		HALT
		←	Mute	
SELECT (wrong parity bit)[f]	SELECT(2) with wrong parity bit	→		HALT
		←	Mute	
AC (empty)	('95 20')	→		READY*(2)
		←	$UIDTX_2$[[1..32]] BCC	

[a] Let $1 \le n_1 \le 32$, $UIDTX_2[[n_1]] = 0$. If such a number does not exist, the test may be skipped.

[b] Let $1 \le n_2 \le 32$, $UIDTX_2[[n_2]] = 1$. If such a number does not exist, the test may be skipped.

[c] Cascade bit of SAK shall be (0)b for double size UID PICCs and (1)b for triple size UID PICCs.

[d] Double size UID PICCs shall be in ACTIVE state; triple size UID PICCs shall be in READY state.

[e] Let $1 \le n_3 \le 32$.

[f] The parity error is simulated on the first transmitted byte of the frame by reversing the parity bit.

Table G.22 *(continued)*

Transition	PICC-test-apparatus		PICC	TTS

AC
(split after (0)b)
	('95' NVB $UIDTX_2[[1..n_1]]$)[a]	→
		←
			if n_1 = 32 then
			(BCC)
			else
			($UIDTX_2[[n_1+1..32]]$ BCC)[a]
				READY*(2)

AC
(split after (1)b)
	('95' NVB $UIDTX_2[[1..n_2]]$)[b]	→
		←
			if n_2 = 32 then
			(BCC)
			else
			($UIDTX_2[[n_2+1..32]]$ BCC)[b]
				READY*(2)

nAC (wrong UID)	('95' NVB ~$UIDTX_2[[1..n_3]]$) [e]	→ ←	Mute	HALT
SELECT	SELECT(2)	→ ←	SAK[c]	ACTIVE or READY[d]
nSELECT (wrong UID)	('95 70' ~$UIDTX_2$ BCC CRC_A)	→ ←	Mute	HALT
Error condition	('95 70' $UIDTX_2$ BCC ~CRC_A)	→ ←	Mute	HALT
ISO/IEC 14443-4 command	$I(0)_0$(TEST_COMMAND1(1))	→ ←	Mute	HALT
DESELECT	S(DESELECT)	→ ←	Mute	HALT
RATS	RATS(0,0)	→ ←	Mute	HALT
PPS	PPS(0,0,0)	→ ←	Mute	HALT

[a] Let $1 \le n_1 \le 32$, $UIDTX_2[[n_1]] = 0$. If such a number does not exist, the test may be skipped.

[b] Let $1 \le n_2 \le 32$, $UIDTX_2[[n_2]] = 1$. If such a number does not exist, the test may be skipped.

[c] Cascade bit of SAK shall be (0)b for double size UID PICCs and (1)b for triple size UID PICCs.

[d] Double size UID PICCs shall be in ACTIVE state; triple size UID PICCs shall be in READY state.

[e] Let $1 \le n_3 \le 32$.

[f] The parity error is simulated on the first transmitted byte of the frame by reversing the parity bit.

G.3.3.10.3 Test report

Fill the appropriate row in Table G.112 according to Table G.23.

Table G.23 — Result criteria for behavior of the PICC Type A in the READY*(2) state

Explanation	Test result
If the PICC has a single size UID	Not applicable (N/A)
When the PICC has a double or triple size UID and only when if responded as indicated in the procedure	PASS
Any other case	FAIL

G.3.3.11 Scenario G.10: Behavior of the PICC Type A in the READY*(3) state

G.3.3.11.1 Scope

This test is to determine the behavior of the PICC Type A in the READY* state of cascade level 3 according to ISO/IEC 14443-3:2018, 6.3.6. This test is only for PICCs with triple size UID.

G.3.3.11.2 Procedure

Perform the following steps for every row of Table G.24:

a) Put the PICC into READY*(3) state.

b) Perform the state transition by sending the command as indicated in the PICC-test-apparatus column.

c) Check if the PICC response is as indicated in the PICC column.

d) Check if the PICC is in the Test Target State (TTS).

Table G.24 — Transitions from READY*(3) state

Transition	PICC-test-apparatus		PICC	TTS
REQA	REQA	→		HALT
		←	Mute	
WUPA	WUPA	→		HALT
		←	Mute	
HLTA	HLTA	→		HALT
		←	Mute	
AC (wrong parity bit)[d]	('97 20', wrong parity bit)	→		HALT
		←	Mute	
SELECT (wrong parity bit)[d]	SELECT(3) with wrong parity bit	→		HALT
		←	Mute	
AC (empty)	('97 20')	→		READY*(3)
		←	$UIDTX_3[[1..32]]$ BCC	
AC (split after (0)b)	('97' NVB $UIDTX_3[[1..n_1]]$)[a]	→		READY*(3)
		←	if $n_1 = 32$ then (BCC) else ($UIDTX_3[[n_1+1..32]]$ BCC)[a]	
AC (split after (1)b)	('97' NVB $UIDTX_3[[1..n_2]]$)[b]	→		READY*(3)
		←	if $n_2 = 32$ then (BCC) else ($UIDTX_3[[n_2+1..32]]$ BCC)[b]	
nAC (wrong UID)	('97' NVB ~$UIDTX_3[[1..n_3]]$)[c]	→		HALT
		←	Mute	
SELECT	SELECT(3)	→		ACTIVE*
		←	SAK (complete)	

[a] Let $1 \le n_1 \le 32$, UIDTX3[[n_1]] = 0. If such a number does not exist, the test may be skipped.

[b] Let $1 \le n_2 \le 32$, UIDTX3[[n_2]] = 1. If such a number does not exist, the test may be skipped.

[c] Let $1 \le n_3 \le 32$.

[d] The parity error is simulated on the first transmitted byte of the frame by reversing the parity bit.

Table G.24 *(continued)*

Transition	PICC-test-apparatus		PICC	TTS
nSELECT (wrong UID)	('97 70' ~$UIDTX_3$ BCC CRC_A)	→		HALT
		←	Mute	
Error condition	('97 70' $UIDTX_3$ BCC ~CRC_A)	→		HALT
		←	Mute	
ISO/IEC 14443-4 command	$I(0)_0$(TEST_COMMAND1(1))	→		HALT
		←	Mute	
DESELECT	S(DESELECT)	→		HALT
		←	Mute	
RATS	RATS(0,0)	→		HALT
		←	Mute	
PPS		→		HALT
		←	Mute	

[a] Let $1 \le n_1 \le 32$, UIDTX3[[n_1]] = 0. If such a number does not exist, the test may be skipped.

[b] Let $1 \le n_2 \le 32$, UIDTX3[[n_2]] = 1. If such a number does not exist, the test may be skipped.

[c] Let $1 \le n_3 \le 32$.

[d] The parity error is simulated on the first transmitted byte of the frame by reversing the parity bit.

G.3.3.11.3 Test report

Fill the appropriate row in Table G.112 according to Table G.25.

Table G.25 — Result criteria for behavior of the PICC Type A in the READY*(3) state

Explanation	Test result
If the PICC has a single or double size UID	Not applicable (N/A)
When the PICC has a triple size UID and only when it responded as indicated in the procedure	PASS
Any other case	FAIL

G.3.3.12 Scenario G.11: Behavior of the PICC Type A in the ACTIVE* state

G.3.3.12.1 Scope

This test is to determine the behavior of the PICC Type A in the ACTIVE* state according to ISO/IEC 14443-3:2018, 6.3.7.

G.3.3.12.2 Procedure

Perform the following steps for every row of Table G.26:

a) Put the PICC into ACTIVE* state.

b) Perform the state transition by sending the command as indicated in the PICC-test-apparatus column.

c) Check if the PICC response is as indicated in the PICC column.

d) Check if the PICC is in the Test Target State (TTS).

Table G.26 — Transitions from ACTIVE* state

Transition	PICC-test-apparatus		PICC	TTS
REQA	REQA	→		HALT
		←	Mute	
WUPA	WUPA	→		HALT
		←	Mute	
HLTA	HLTA	→		HALT
		←	Mute	
AC	('93' NVB UIDTX$_1$[[1..n$_1$]])[a]	→		HALT
		←	Mute[b]	
nAC	('93' NVB ~UIDTX$_1$[[1..n$_1$]])[a]	→		HALT
		←	Mute[b]	
SELECT	SELECT(1)	→		HALT
		←	Mute[b]	
nSELECT	('93 70' ~UIDTX$_1$ BCC CRC_A)	→		HALT
		←	Mute[b]	
RATS	RATS(0,0)	→		PROTOCOL
		←	ATS	
Error condition	('E0 00' ~CRC_A)	→		HALT
		←	Mute[b]	
ISO/IEC 14443-4 command	I(0)$_0$(TEST_COMMAND1(1))	→		HALT
		←	Mute[b]	
DESELECT	S(DESELECT)	→		HALT
		←	Mute[b]	
PPS	PPS(0,0,0)	→		HALT
		←	Mute[b]	
RATS (wrong parity bit)[c]	RATS(0,0) with wrong parity bit	→		HALT
		←	Mute[b]	
Type B command	REQB	→		HALT or ACTIVE*[d]
		←	Mute	
AC (empty)	('93 20')	→		HALT
		←	Mute	

[a] Let $1 \le n_1 \le 32$.

[b] Mute or proprietary response.

[c] The parity error is simulated on the first transmitted byte of the frame by reversing the parity bit.

[d] Check first HALT state as TTS. If TTS is not HALT state, rerun the test a second time to check ACTIVE* state.

G.3.3.12.3 Test report

Fill the appropriate row in Table G.112 according to Table G.27.

Table G.27 — Result criteria for behavior of the PICC Type A in the ACTIVE* state

Explanation	Test result
Only when the PICC responded as indicated in the procedure	PASS

Table G.27 *(continued)*

Explanation	Test result
Any other case	FAIL

G.3.3.13 Scenario G.12: Behavior of the PICC Type A in the PROTOCOL state

G.3.3.13.1 Scope

This test is to determine the behavior of the PICC Type A in the PROTOCOL state according to ISO/IEC 14443-3:2018, 6.3.8. This test shall ensure that the activated PICC does not respond to any anticollision or initialization command.

G.3.3.13.2 Procedure

Perform the following steps for every row of Table G.28:

a) Put the PICC into PROTOCOL state.

b) Perform the state transition by sending the command as indicated in the PICC-test-apparatus column.

c) Check if the PICC response is as indicated in the PICC column.

d) Check if the PICC is in the Test Target State (TTS).

Table G.28 — Transitions from PROTOCOL state

Transition	PICC-test-apparatus		PICC	TTS
REQA	REQA	→		PROTOCOL
		←	Mute	
WUPA	WUPA	→		PROTOCOL
		←	Mute	
AC	('93' NVB $UIDTX_1[[1..n_1]]$)[a]	→		PROTOCOL
		←	Mute	
nAC	('93' NVB ~$UIDTX_1[[1..n_1]]$)[a]	→		PROTOCOL
		←	Mute	
HLTA	HLTA	→		PROTOCOL
		←	Mute	
SELECT	SELECT(1)	→		PROTOCOL
		←	Mute	
nSELECT	('93 70' ~$UIDTX_1$ BCC CRC_A)	→		PROTOCOL
		←	Mute	
RATS	RATS(0,0)	→		PROTOCOL
		←	Mute	
Error condition	S(DESELECT, ~CRC_A)	→		PROTOCOL
		←	Mute	

[a] Let $1 \le n_1 \le 32$.

[b] PPS response is returned if the PICC supports PPS.

[c] The parity error is simulated on the first transmitted byte of the frame by reversing the parity bit.

[d] Check first IDLE state as TTS. If TTS is not IDLE state, rerun the test a second time to check PROTOCOL state.

Table G.28 *(continued)*

Transition	PICC-test-apparatus		PICC	TTS
DESELECT	S(DESELECT)	→		HALT
		←	S(DESELECT)	
PPS	PPS(0,0,0)	→		PROTOCOL
		←	Mute, or PPS response[b]	
ISO/IEC 14443-4 command	$I(0)_0$(TEST_COMMAND1(1))	→		PROTOCOL
		←	$I(0)_0$ (TEST_RESPONSE1(1))	
DESELECT (wrong parity bit)[c]	S(DESELECT) with wrong parity bit	→		PROTOCOL
		←	Mute	
ISO/IEC 14443-4 command (wrong parity bit)[c]	$I(0)_0$(TEST_COMMAND1(1)) with wrong parity bit	→		PROTOCOL
		←	Mute	
Type B command	REQB	→		IDLE or PROTOCOL[d]
		←	Mute	
AC (empty)	('93 20')	→		PROTOCOL
		←	Mute	

[a] Let $1 \leq n_1 \leq 32$.

[b] PPS response is returned if the PICC supports PPS.

[c] The parity error is simulated on the first transmitted byte of the frame by reversing the parity bit.

[d] Check first IDLE state as TTS. If TTS is not IDLE state, rerun the test a second time to check PROTOCOL state.

G.3.3.13.3 Test report

Fill the appropriate row in Table G.112 according to Table G.29.

Table G.29 — Result criteria for behavior of the PICC Type A in the PROTOCOL state

Explanation	Test result
Only when the PICC responded as indicated in the procedure	PASS
Any other case	FAIL

G.3.4 Scenario G.13: Handling of Type A anticollision

G.3.4.1 Scope

This test is to perform a full bitwise anticollision loop according to ISO/IEC 14443-3:2018, 6.5.3.

G.3.4.2 Procedure

a) Put the PICC into the field.

b) Put the PICC into READY(1) state.

c) Execute AnticollisionA.

d) Put the PICC into READY*(1) state.

e) Execute AnticollisionA.

Pseudo code: Type A anticollision procedure

```
1     Procedure AnticollisionA
2     // TPDUSend and TPDURecv are PCD specific functions
3     // to send and receive frames
4     for c = 1 to CascadeLevels do
5       TPDUSend (SEL(c) 20)
6       if TPDURecv() ≠ (UIDTXc[[1..32]] BCC) then return FAIL
7       // anticollision over UID bits
8       for p = 1 to 32 do
9         // enter desired cascade level
10        if c ≥ 2 then TPDUSend(SELECT(1))
11        if c = 3 then TPDUSend(SELECT(2))
12        // anticollision with matched bit
13        NVB[[1..4]] = (p + 16) mod 8
14        NVB[[5..8]] = (p + 16) div 8
15        TPDUSend (SEL(c) NVB UIDTXc[[1..p]])
16        if TPDURecv() ≠ (UIDTXc[[p+1..32]] BCC) then return FAIL
17        // anticollision with unmatched bit
18        TPDUSend(SEL(c) NVB UIDTXc[[1..p-1]] ~UIDTXc[[p]])
19        if TPDURecv() ≠ Mute then return FAIL
20        // re-enter READY(1) (resp. READY*(1)) state
21        TPDUSend (WUPA)
22      end for
23    end for
24    return PASS
```

G.3.4.3 Test report

Fill the appropriate row in Table G.112 according to Table G.30.

Table G.30 — Result criteria for handling of Type A anticollision

Explanation	Test result
Only when every Anticollision Test procedure has returned PASS	PASS
When any Anticollision Test procedure has returned the value FAIL	FAIL

G.3.5 Handling of RATS

Handling of RATS is tested in G.3.3.7 and G.3.3.12.

Scenario G.14: Void

Scenario G.14: RATS after wrong RATS, which was defined in former editions of this document, is no longer present.

Scenario G.15: Void

Scenario G.15: RATS after RATS, which was defined in former editions of this document, is no longer present.

G.3.6 Handling of PPS request

G.3.6.1 Scope

This test is to determine the handling of the PPS request by the PICC Type A according to ISO/IEC 14443-4: 2018, 5.6.2.2.

G.3.6.2 Procedure

For each of the scenarios described in Table G.31, Table G.32 and Table G.33, perform the following steps:

a) Put the PICC into PROTOCOL state.

b) Send the command as described under the PICC-test-apparatus column in each scenario.

c) Check that the response of the PICC conforms to the one given in the PICC column.

d) Check if the PICC is in PROTOCOL state.

Scenario G.16: Void

Scenario G.16: PPS without parameter change, which was defined in former editions of this document, is no longer present.

Table G.31 — Scenario G.17: PPS without PPS1

PICC-test-apparatus		PICC
('D0 01' CRC_A)	→	
	←	Mute or ('D0' CRC_A)[a]
[a] Both responses are valid.		

Table G.32 — Scenario G.18: PPS after PPS

PICC-test-apparatus		PICC
PPS(0,0,0)	→	
	←	Mute or ('D0' CRC_A)[a]
PPS(0,0,0)	→	
	←	Mute
[a] Response depends on whether the PICC supports PPS or not. If the PICC does not support any changeable parameters it may not support the PPS request because the PCD shall not send PPS to such a PICC (see ISO/IEC 14443-4:2018, Clause 5, 6th dash).		

Table G.33 — Scenario G.19: PPS after unreceived PPS

PICC-test-apparatus		PICC
('D0 11 00' ~CRC_A)	→	
	←	Mute
PPS(0,0,0)	→	
	←	Mute

G.3.6.3 Test report

Fill the appropriate rows in Table G.112 according to Table G.34.

Table G.34 — Result criteria for handling of PPS request

Explanation	Test result
Only when the PICC responded as indicated in the procedure	PASS
Any other case	FAIL

G.3.7 Scenario G.20: Handling of FSD

G.3.7.1 Scope

This test is to determine if the PICC Type A respects the FSD value as negotiated by the RATS according to ISO/IEC 14443-4:2018, 5.1.

G.3.7.2 Procedure

Perform the following steps for each FSDI defined in ISO/IEC 14443-4:

a) Put the PICC into ACTIVE state.

b) Send the RATS(0, fsdi) command with parameter fsdi as in the particular test.

c) Check that the PICC answer is a valid ATS and that its size is ≤ FSD.

NOTE The PICC may require additional sequences to be ready to accept TEST_COMMAND2(2).

d) Send the I-block $I(0)_0$(TEST_COMMAND2(2)).

e) Check that the size of the I-block sent by the PICC is ≤ FSD.

G.3.7.3 Test report

Fill the appropriate row in Table G.112 according to Table G.35.

Table G.35 — Result criteria for Scenario G.20: Handling of FSD

Explanation	Test result
Only when the PICC responded as indicated in the procedure	PASS
Any other case	FAIL

G.3.8 Handling of Frame Delay Time PICC to PCD and SFGT

G.3.8.1 Scope

This test is to determine if the PICC Type A respects the minimum Frame Delay Time PICC to PCD defined in ISO/IEC14443-3:2018, 6.2.1.2 and SFGT defined in ISO/IEC 14443-4:2018, 5.2.5.

G.3.8.2 Procedure

This procedure shall be repeated for two combinations of bit rates as described below:

1) at a bit rate of $f_c/128$ in both directions;

2) at the maximum bit rates supported by the PICC in each direction.

Perform the following steps for Scenario G.71:

a) Put the PICC into IDLE state.

b) For each step in the scenario do:

1) Send the command as described in the PICC-test-apparatus column respecting the timing condition described in the Frame Delay Time PICC to PCD column.

2) Check that the PICC response matches the one of the PICC column.

c) End for.

Table G.36 — Scenario G.71: Handling of Frame Delay Time PICC to PCD and SFGT

Step	Frame Delay Time PICC to PCD	PICC-test-apparatus		PICC
1	—	REQA	→	
			←	ATQA
2	Minimum Frame Delay Time PICC to PCD + $50/f_c$[a]	AC(1)	→	
			←	$UIDTX_1$ BCC
3	$(1172 + 50)/f_c$[a]	SELECT(1)	→	
			←	SAK (cascade) or SAK (complete)
4[b]	$(1172 + 50)/f_c$[a]	AC(2)	→	
			←	$UIDTX_2$ BCC
5[b]	$(1172 + 50)/f_c$[a]	SELECT(2)	→	
			←	SAK (cascade) or SAK (complete)
6[c]	$(1172 + 50)/f_c$[a]	AC(3)	→	
			←	$UIDTX_3$ BCC
7[c]	$(1172 + 50)/f_c$[a]	SELECT(3)	→	
			←	SAK (complete)
8	$(1172 + 50)/f_c$[a]	RATS(0,0)	→	
			←	ATS
9	$(1172 + 50)/f_c$ if SFGI=0 or SFGT + $50/f_c$[a]	PPS(0,dri,dsi)[d]	→	
			←	PPS response
10	$(1172 + 50)/f_c$ if SFGI=0 or SFGT + $50/f_c$[a,e]	$I(0)_0$(TEST_COMMAND1(1))	→	
			←	$I(0)_0$(TEST_RESPONSE1(1))[f]
11	$(1172 + 50)/f_c$[a]	$I(0)_1$(TEST_COMMAND3)	→	
			←	S(WTX)(WTXM)
12	$(1172 + 50)/f_c$[a]	S(WTX)(WTXM)	→	
			←	$I(0)_1$(TEST_RESPONSE3)[g]

[a] The applied Frame Delay Time PICC to PCD shall have a maximum tolerance of $\pm 25/f_c$.

[b] This step is sent only for a PICC with UID size 2 or 3.

[c] This step is sent only for a PICC with UID size 3.

[d] PPS request is sent for bit rate > $f_c/128$. Configure PPS command with appropriate dsi and dri corresponding to the maximum bit rates supported by the PICC in each direction. Skip this step if the PICC does not support any changeable parameters in the ATS.

[e] If the previous step has been skipped since the PICC does not support any changeable parameters in the ATS, SFGI applies to this step.

[f] S(WTX) request(s) may be sent by the PICC. The PICC-test-apparatus shall acknowledge each S(WTX) request with an S(WTX) response until receiving the PICC response. There is no specific requirement on Frame Delay Time PICC to PCD for the S(WTX) response(s).

[g] Additional S(WTX) request(s) may be sent by the PICC. The PICC-test-apparatus shall acknowledge each S(WTX) request with an S(WTX) response until receiving the PICC response. There is no specific requirement on Frame Delay Time PICC to PCD for the S(WTX) response(s).

G.3.8.3 Test report

Fill the appropriate row in Table G.112 according to Table G.37.

Table G.37 — Result criteria for handling of Frame Delay Time PICC to PCD

Explanation	Test result
Only when the PICC responded as indicated in the procedure	PASS
Any other case	FAIL

G.3.9 Scenario G.72: PICC bit rates capability

G.3.9.1 Scope

This test is to determine the behavior of a Type A PICC when receiving PCD messages according to ISO/IEC 14443-3:2018, 6.1 and 6.2.

G.3.9.2 Procedure

This procedure shall be repeated for combinations of PCD to PICC and PICC to PCD bit rates supported by the PICC, so that all supported bit rates in each direction are used at least once during the tests.

a) Put the PICC into IDLE state.

b) Put the PICC in PROTOCOL state

c) Send the appropriate PPS command and check the PPS response.

d) Check that the PICC responses are valid. Record the presence, content and timing of the PICC responses.

e) Send the I-block $I(0)_0$(TEST_COMMAND1(1)). Record the presence, frame format, content and timing of the PICC response.

f) Send the S(DESELECT) command.

g) Check that the PICC response is a valid S(DESELECT). Record the presence, frame format, content and timing of the PICC response.

h) Send a WUPA command.

i) Check that the PICC response is a valid ATQA. Record the presence, frame format, content and timing of the PICC response.

G.3.9.3 Test report

Report the measured timings in Table G.110 and fill the appropriate row in Table G.112 according to Table G.38.

Table G.38 — Result criteria for Scenario G.72: PICC bit rates capability

Explanation	Test result
Only when the PICC responded as indicated in the procedure	PASS
Any other case	FAIL

G.4 Test method for initialization of the PICC Type B

G.4.1 General

This subclause is to test if the PICC Type B conforms to ISO/IEC 14443-3. If compliance with ISO/IEC 14443-4 is not required all tests containing ISO/IEC 14443-4 commands need not be applied.

If the PICC does not support the REQB/WUPB with N > 1 nor the Slot-MARKER command (see ISO/IEC 14443-3:2018, 7.6.1) all tests containing these commands need not be applied.

G.4.2 Scenario G.21: Polling

G.4.2.1 Scope

This test is to determine the behavior of the PICC Type B on receiving of REQB according to ISO/IEC 14443-3:2018, 5.3.

G.4.2.2 Procedure

Perform the following steps for 3 different operating field strengths of H_{min}, $(H_{min} + H_{max})/2$ and H_{max} as specified for the respective PICC class:

a) Place the PICC into the field and adjust it.

b) Switch the RF operating field off for a minimum time for resetting a PICC in accordance with ISO/IEC 14443-3:2018, 5.6.

c) Switch the RF operating field on.

d) Wait 5 ms and send a valid REQB(1) command frame.

e) Record the presence and the content of the PICC response.

f) Switch the RF operating field off for a minimum time for resetting a PICC in accordance with ISO/IEC 14443-3:2018, 5.6.

g) Switch the RF operating field on.

h) Wait 5 ms and send a valid REQA command frame (with Type A modulation).

i) Wait 5 ms and send a valid REQB(1) command frame.

j) Record the presence and the content of the PICC response.

G.4.2.3 Test report

Fill the appropriate row in Table G.113 according to Table G.39:

Table G.39 — Result criteria for Scenario G.21: Polling

Explanation	Test result
Only when the PICC's response is a valid ATQB in both steps e) and j)	PASS
When the PICC's response isn't a valid ATQB in any of steps e) or j)	FAIL

G.4.3 Scenario G.22: PICC framing and bit rates capability

G.4.3.1 Scope

This test is to determine the behavior of a Type B PICC when receiving PCD messages according to ISO/IEC 14443-3:2018, 7.1.

G.4.3.2 Procedure

This procedure shall be repeated for combinations of PCD to PICC and PICC to PCD bit rates supported by the PICC, so that all supported bit rates in each direction are used at least once during the tests.

Perform the following steps for each row of Table G.40:

a) Put the PICC into IDLE state.

b) Set the frame parameters of the PICC-test-apparatus according to Table G.36.

c) Send a REQB command.

d) Check that the PICC response is a valid ATQB. Record the presence, frame format, content and timing of the PICC response.

e) Send the ATTRIB(0,0).

f) Check that the PICC response is a valid Answer to ATTRIB(0). Record the presence, frame format, content and timing of the PICC response.

g) Send an S(PARAMETERS) block to send Bit rates Request.

h) Check PICC answer. If the PICC supports S(PARAMETERS) blocks, the PICC responds with an S(PARAMETERS) block containing values for all supported parameters. If the PICC does not support S(PARAMETERS) it stays mute; skip steps i) and j).

i) Send an S(PARAMETERS) block to activate selected communication parameters (bit rates and framing options).

j) Check that the PICC response is a valid S(PARAMETERS) block to acknowledge the activated parameters.

k) Send the I-block $I(0)_0$(TEST_COMMAND1(1)). Record the presence, frame format, content and timing of the PICC response.

l) Send the S(DESELECT) command.

m) Check that the PICC response is a valid S(DESELECT). Record the presence, frame format, content and timing of the PICC response.

n) Send a WUPB command.

o) Check that the PICC response is a valid ATQB. Record the presence, frame format, content and timing of the PICC response.

Table G.40 — Type B PCD to PICC frame parameters

Test No.	EGT [etu]	SOF low [etu]	SOF high [etu]	EOF [etu]	Min. TR0 coding	Min. TR1 coding	EOF handling coding	SOF handling coding	Framing options (b2b1)[a]	Bit boundaries[b]	
										Rising edge [$1/f_c$]	Falling edge [$1/f_c$]
1	0	10,5	2,5	10,5	(00)b	(00)b	(0)b	(0)b	(00)b	0	0
2	0	10 – 1/16	2 – 1/8	10 – 1/16	(00)b	(00)b	(0)b	(0)b	(00)b	0	0
3	6[c]	10 – 1/16	2 – 1/8	10 – 1/16	(00)b	(00)b	(0)b	(0)b	(00)b	0	0
4	0	11 + 1/8	2 – 1/8	10 – 1/16	(00)b	(00)b	(0)b	(0)b	(00)b	0	0
5	0	10 – 1/16	3 + 1/8	10 – 1/16	(00)b	(00)b	(0)b	(0)b	(00)b	0	0
6	6[c]	11 + 1/8	3 + 1/8	11 + 1/8	(00)b	(00)b	(0)b	(0)b	(00)b	0	0
7	0	10 – 1/16	2 – 1/8	10 – 1/16	(00)b	(00)b	(0)b	(0)b	(00)b	–8[d]	–8[e]
8	0	10 – 1/16	2 – 1/8	10 – 1/16	(00)b	(00)b	(0)b	(0)b	(00)b	+8[f]	+8[g]
9	0	10 – 1/16	2 – 1/8	10 – 1/16	(00)b	(00)b	(0)b	(0)b	(00)b	–8[d]	+8[g]
10	0	10 – 1/16	2 – 1/8	10 – 1/16	(00)b	(00)b	(0)b	(0)b	(00)b	+8[f]	–8[e]
11	0	10,5	2,5	10,5	(00)b	(01)b	(0)b	(0)b	(00)b	0	0
12	0	10,5	2,5	10,5	(00)b	(10)b	(0)b	(0)b	(00)b	0	0
13	0	10,5	2,5	10,5	(01)b	(00)b	(0)b	(0)b	(00)b	0	0
14	0	10,5	2,5	10,5	(01)b	(01)b	(0)b	(0)b	(00)b	0	0
15	0	10,5	2,5	10,5	(01)b	(10)b	(0)b	(0)b	(00)b	0	0
16	0	10,5	2,5	10,5	(10)b	(00)b	(0)b	(0)b	(00)b	0	0
17	0	10,5	2,5	10,5	(10)b	(01)b	(0)b	(0)b	(00)b	0	0
18	0	10,5	2,5	10,5	(10)b	(10)b	(0)b	(0)b	(00)b	0	0
19	0	10,5	2,5	10,5	(00)b	(00)b	(0)b	(1)b	(00)b	0	0
20	0	10,5	2,5	10,5	(00)b	(00)b	(1)b	(0)b	(00)b	0	0
21	0	10,5	2,5	10,5	(00)b	(00)b	(1)b	(1)b	(00)b	0	0
22	0	10,5	2,5	10,5	(00)b	(00)b	(0)b	(0)b	(01)b	0	0
23	0	10,5	2,5	10,5	(00)b	(00)b	(0)b	(0)b	(10)b	0	0

[a] Only applicable if PICC supports at least one framing options.

[b] Bit boundaries is applied on b2 of the first byte of I-block $I(0)_0$(TEST_COMMAND1(1)). For bit rates higher than $f_c/16$, bit boundaries shall occur at nominal bit positions.

[c] Not applicable for bit rates higher that $f_c/16$.

[d] Apply -4 for $f_c/64$, -2 for $f_c/32$ and –1 for $f_c/16$.

[e] Apply –1 for $f_c/64$, $f_c/32$ and for $f_c/16$.

[f] Apply +4 for $f_c/64$, +2 for $f_c/32$ and +1 for $f_c/16$.

[g] Apply +1 for $f_c/64$, $f_c/32$ and for $f_c/16$.

G.4.3.3 Test report

Report the measured timings in Table G.111 and fill the appropriate row in Table G.113 according to Table G.41.

Table G.41 — Result criteria for Scenario G.22: PICC Reception

Explanation	Test result
Only when the PICC responded as indicated in the procedure	PASS
Any other case	FAIL

G.4.4 Testing of the PICC Type B state transitions

G.4.4.1 General

These tests are to verify the correct implementation of the PICC Type B state diagram as described in ISO/IEC 14443-3:2018, 7.4.

G.4.4.2 General test outline

G.4.4.2.1 General

This is the same procedure as described for the PICC Type A (see G.3.3.2).

G.4.4.2.2 Functions to set the PICC in Test Initial State (TIS)

Putting the PICC into the Test Initial State (TIS) will be done by a sequence of transition commands specified in the following tables. The general method is as follows:

In order to put the PICC into the Test Initial State (TIS), look up the corresponding sequence in Table G.42. Then successively apply the state transitions described in this column by looking up the corresponding commands in Table G.43. Always check the content and integrity of the PICC response.

Table G.42 — State transition sequence

TIS	State Transition Sequence
POWER-OFF	—
IDLE[a]	POWER-OFF → IDLE
READY-REQUESTED	POWER-OFF → IDLE → READY-REQUESTED
READY-DECLARED	POWER-OFF → IDLE → READY-DECLARED
PROTOCOL	POWER-OFF → IDLE → READY-DECLARED → PROTOCOL
HALT	POWER-OFF → IDLE → READY-DECLARED → HALT
[a] IDLE state may be reached from READY-DECLARED state.	

Table G.43 — State transition

State → Next State	PICC-test-apparatus		PICC
POWER-OFF → IDLE	Power On (RF operating field on)	→	
		←	Mute
IDLE → READY-REQUESTED	REQB(16)	→	
		←	Mute[a]
IDLE → READY-DECLARED	REQB(1)	→	
		←	ATQB
READY-DECLARED → HALT	HLTB	→	
		←	'00' CRC_B
READY-DECLARED → PROTOCOL	ATTRIB(0,0)	→	
		←	Answer to ATTRIB(0)
READY-DECLARED → IDLE	REQB(1,nAFI)	→	
		←	Mute
[a] In case the PICC has selected slot 1, the REQB command shall be reissued until the PICC doesn't answer ATQB.			

G.4.4.2.3 Functions for checking the validity of the Test Target State (TTS)

Table G.44 describes the state transitions, which are used to check whether the PICC is in the state S. The content of the PICC answer (i.e. ATQB...) should be thoroughly checked for ISO/IEC 14443-3 and ISO/IEC 14443-4 conformance.

NOTE 1 The tests can cause the PICC to change its state.

Table G.44 — Checking the TTS

TTS	PICC-test-apparatus		PICC
IDLE	REQB(1)	→	
		←	ATQB[b]
READY-REQUESTED	SLOTMARKER (n)[a]	→	
		←	ATQB[b]
READY-DECLARED	ATTRIB(0,0)	→	
		←	Answer to ATTRIB(0)
PROTOCOL	$I(0)_0$(TEST_COMMAND1(1))	→	
		←	$I(0)_0$(TEST_RESPONSE1(1))
HALT	REQB(1)	→	
		←	Mute
	WUPB(1)	→	
		←	ATQB[b]

[a] Since the selected PICC slot is unknown, the Slot-MARKER command shall be reissued with different slot values until an ATQB is received.

[b] If the PUPI is known, check that PUPI has not changed.

NOTE 2 The block number can be 0 or 1 dependent on block numbering rules, see ISO/IEC 14443-4:2018, 7.5.4.

G.4.4.3 Scenario G.23: Behavior of the PICC Type B in the IDLE state

G.4.4.3.1 Scope

This test is to determine the behavior of the PICC Type B in the IDLE state according to ISO/IEC 14443-3:2018, 7.4.4.

G.4.4.3.2 Procedure

Perform the following steps for every row of Table G.45:

a) Put the PICC into IDLE state. If the PUPI is unknown, put the PICC in READY-DECLARED state to retrieve its PUPI and put it in IDLE state using READY-DECLARED → IDLE state transition.

b) Perform the state transition by sending the command as indicated in the PICC-test-apparatus column.

c) Check if the PICC response is as indicated in the PICC column.

d) Check if the PICC is in the Test Target State (TTS).

Table G.45 — Transitions from IDLE state

Transition	PICC-test-apparatus		PICC	TTS
REQB[d]	REQB(1)	→		READY-DECLARED
		←	ATQB	
WUPB[d]	WUPB(1)	→		READY-DECLARED
		←	ATQB	
REQB (wrong CRC)	('05 00 00' ~ CRC_B)	→		IDLE
		←	Mute	
WUPB (wrong CRC)	('05 00 08' ~ CRC_B)	→		IDLE
		←	Mute	
HLTB	HLTB	→		IDLE
		←	Mute	
ATTRIB	ATTRIB(0,0)	→		IDLE
		←	Mute	
Slot-MARKER	SLOTMARKER(n)[a]	→		IDLE
		←	Mute	
ISO/IEC 14443-4 command	$I(0)_0$(TEST_COMMAND1(1))	→		IDLE
		←	Mute	
DESELECT	S(DESELECT)	→		IDLE
		←	Mute	
REQB[e] (Unmatched AFI)	REQB(1,nAFI)	→		IDLE
		←	Mute	
WUPB[e] (Unmatched AFI)	WUPB(1,nAFI)	→		IDLE
		←	Mute	
HLTB (Unmatched PUPI)	HLTB(~PUPI)	→		IDLE
		←	Mute	
ATTRIB (Unmatched PUPI)	ATTRIB(0, 0, ~PUPI)	→		IDLE
		←	Mute	
REQB[c]	REQB(16)[b]	→		READY-REQUESTED
		←	Mute	
WUPB[c]	WUPB(16)[b]	→		READY-REQUESTED
		←	Mute	

[a] n shall run through all values $2 \leq n \leq 16$.

[b] Nevertheless, there is statistically a probability of 1/16 so that the PICC answers ATQB and goes to READY-DECLARED sub-state.

[c] If the PICC does not support the REQB/WUPB with N > 1 (see ISO/IEC 14443-3:2018, 7.6.1) the test need not be applied.

[d] All matched AFIs as defined by the PICC manufacturer shall be tested.

[e] Any unmatched AFI values, also values which are currently RFU, should be tested.

G.4.4.3.3 Test report

Fill the appropriate row in Table G.113 according to Table G.46.

Table G.46 — Result criteria for Scenario G.23: Behavior of the PICC Type B in the IDLE state

Explanation	Test result
Only when the PICC responded as indicated in the procedure	PASS
Any other case	FAIL

G.4.4.4 Scenario G.24: Behavior of the PICC Type B in the READY-REQUESTED sub-state

G.4.4.4.1 General

If the PICC does not support the REQB/WUPB with N > 1 nor the Slot-MARKER command (see ISO/IEC 14443-3:2018, 7.6.1) this scenario need not be applied.

G.4.4.4.2 Scope

This test is to determine the behavior of the PICC Type B in the READY-REQUESTED sub-state according to ISO/IEC 14443-3:2018, 7.4.5.

G.4.4.4.3 Procedure

Perform the following steps for every row of Table G.47:

a) Put the PICC into READY-REQUESTED state. If the PUPI is unknown, put the PICC in READY-DECLARED state to retrieve the PICC PUPI and put it in IDLE state before performing the step a).

b) Perform the state transition by sending the command as indicated in the PICC-test-apparatus column.

c) Check if the PICC response is as indicated in the PICC column.

d) Check if the PICC is in the Test Target State (TTS).

Table G.47 — Transitions from READY-REQUESTED sub-state

Transition	PICC-test-apparatus		PICC	TTS
REQB[d]	REQB(1)	→		READY-DECLARED
		←	ATQB	
WUPB[d]	WUPB(1)	→		READY-DECLARED
		←	ATQB	
REQB (wrong CRC)	('05 00 00' ~ CRC_B)	→		READY-REQUESTED
		←	Mute	
WUPB (wrong CRC)	('05 00 08' ~ CRC_B)	→		READY-REQUESTED
		←	Mute	
HLTB	HLTB	→		READY-REQUESTED
		←	Mute	
ATTRIB	ATTRIB(0,0)	→		READY-REQUESTED
		←	Mute	

[a] n shall run through all values 2 ≤ n ≤ 16. The PICC shall respond ATQB at exactly one value of n, else Mute.

[b] Nevertheless, there is statistically a probability of 1/16 so that the PICC answers ATQB and goes to READY-DECLARED sub-state.

[c] Send all Slot-MARKER commands and verify that there is no response before checking the IDLE state.

[d] All matched AFIs as defined by the PICC manufacturer shall be tested.

[e] Any unmatched AFI values, also values which are currently RFU, should be tested.

Table G.47 *(continued)*

Transition	PICC-test-apparatus		PICC	TTS
Slot-MARKER	SLOTMARKER(n)[a]	→		READY-DECLARED
		←	ATQB or Mute	
ISO/IEC 14443-4 command	$I(0)_0$(TEST_COMMAND1(1))	→		READY-REQUESTED
		←	Mute	
DESELECT	S(DESELECT)	→		READY-REQUESTED
		←	Mute	
REQB	REQB(16)[b]	→		READY-REQUESTED
		←	Mute	
WUPB	WUPB(16)[b]	→		READY-REQUESTED
		←	Mute	
REQB[e] (unmatched AFI)	REQB(1,nAFI)	→		IDLE[c]
		←	Mute	
WUPB[e] (unmatched AFI)	WUPB(1,nAFI)	→		IDLE[c]
		←	Mute	
HLTB (unmatched PUPI)	HLTB(~PUPI)	→		READY-REQUESTED
		←	Mute	
ATTRIB (unmatched PUPI)	ATTRIB(0, 0, ~PUPI)	→		READY-REQUESTED
		←	Mute	

[a] n shall run through all values 2 ≤ n ≤ 16. The PICC shall respond ATQB at exactly one value of n, else Mute.

[b] Nevertheless, there is statistically a probability of 1/16 so that the PICC answers ATQB and goes to READY-DECLARED sub-state.

[c] Send all Slot-MARKER commands and verify that there is no response before checking the IDLE state.

[d] All matched AFIs as defined by the PICC manufacturer shall be tested.

[e] Any unmatched AFI values, also values which are currently RFU, should be tested.

G.4.4.4.4 Test report

Fill the appropriate row in Table G.113 according to Table G.48.

Table G.48 — Result criteria for Scenario G.24: Behavior of the PICC Type B in the READY-REQUESTED sub-state

Explanation	Test result
Only when the PICC responded as indicated in the procedure	PASS
Any other case	FAIL

G.4.4.5 Scenario G.25: Behavior of the PICC Type B in the READY-DECLARED sub-state

G.4.4.5.1 Scope

This test is to determine the behavior of the PICC Type B in the READY-DECLARED sub-state according to ISO/IEC 14443-3:2018, 7.4.6.

G.4.4.5.2 Procedure

Perform the following steps for every row of Table G.49:

a) Put the PICC into READY-DECLARED sub-state.

b) Perform the state transition by sending the command as indicated in the PICC-test-apparatus column.

c) Check if the PICC response is as indicated in the PICC column.

d) Check if the PICC is in the Test Target State (TTS).

Table G.49 — Transitions from READY-DECLARED sub-state

Transition	PICC-test-apparatus		PICC	TTS
REQB[e]	REQB(1)	→		READY-DECLARED
		←	ATQB	
WUPB[e]	WUPB(1)	→		READY-DECLARED
		←	ATQB	
REQB (wrong CRC)	('05 00 00' ~ CRC_B)	→		READY-DECLARED
		←	Mute	
WUPB (wrong CRC)	('05 00 08' ~ CRC_B)	→		READY-DECLARED
		←	Mute	
HLTB	HLTB	→		HALT
		←	('00' CRC_B)	
ATTRIB	ATTRIB(0,0)	→		PROTOCOL
		←	Answer to ATTRIB(0)	
Slot-MARKER	SLOTMARKER(n)[a]	→		READY-DECLARED
		←	Mute	
ISO/IEC 14443-4 command	$I(0)_0$(TEST_COMMAND1(1))	→		READY-DECLARED
		←	Mute	
DESELECT	S(DESELECT)	→		READY-DECLARED
		←	Mute	
REQB[b]	REQB(16)[c]	→		READY-REQUESTED
		←	Mute	
WUPB[b]	WUPB(16)[c]	→		READY-REQUESTED
		←	Mute	
HLTB (unmatched PUPI)	HLTB(~PUPI)	→		READY-DECLARED
		←	Mute	
ATTRIB (unmatched PUPI)	ATTRIB(0, 0, ~PUPI)	→		READY-DECLARED
		←	Mute	

[a] n shall run through all values 2 ≤ n ≤ 16.

[b] If the PICC does not support the REQB/WUPB with N > 1 (see ISO/IEC 14443-3:2018, 7.6.1) the test need not be applied.

[c] Nevertheless, there is statistically a probability of 1/16 so that the PICC answers ATQB and goes to READY-DECLARED sub-state.

[d] Send ATTRIB command and verify that there is no response before checking the IDLE state.

[e] All matched AFIs as defined by the PICC manufacturer shall be tested.

[f] Any unmatched AFI values, also values which are currently RFU, should be tested.

Table G.49 *(continued)*

Transition	PICC-test-apparatus		PICC	TTS
REQB[f] (unmatched AFI)	REQB(1,nAFI)	→		IDLE[d]
		←	Mute	
WUPB[f] (Unmatched AFI)	WUPB(1,nAFI)	→		IDLE[d]
		←	Mute	

[a] n shall run through all values $2 \leq n \leq 16$.

[b] If the PICC does not support the REQB/WUPB with N > 1 (see ISO/IEC 14443-3:2018, 7.6.1) the test need not be applied.

[c] Nevertheless, there is statistically a probability of 1/16 so that the PICC answers ATQB and goes to READY-DECLARED sub-state.

[d] Send ATTRIB command and verify that there is no response before checking the IDLE state.

[e] All matched AFIs as defined by the PICC manufacturer shall be tested.

[f] Any unmatched AFI values, also values which are currently RFU, should be tested.

G.4.4.5.3 Test report

Fill the appropriate row in Table G.113 according to Table G.50.

Table G.50 — Result criteria for Scenario G.25: Behavior of the PICC Type B in the READY-DECLARED sub-state

Explanation	Test result
Only when the PICC responded as indicated in the procedure	PASS
Any other case	FAIL

G.4.4.6 Scenario G.26: Behavior of the PICC Type B in the HALT state

G.4.4.6.1 Scope

This test is to determine the behavior of the PICC Type B in the HALT state according to ISO/IEC 14443-3:2018, 7.4.8.

G.4.4.6.2 Procedure

Perform the following steps for every row of Table G.51:

a) Put the PICC into HALT state.

b) Perform the state transition by sending the command as indicated in the PICC-test-apparatus column.

c) Check if the PICC response is as indicated in the PICC column.

d) Check if the PICC is in the Test Target State (TTS).

Table G.51 — Transitions from HALT state

Transition	PICC-test-apparatus		PICC	TTS
REQB[d]	REQB(1)	→		HALT
		←	Mute	
WUPB[d]	WUPB(1)	→		READY-DECLARED
		←	ATQB	
WUPB (wrong CRC)	('05 00 08' ~ CRC_B)	→		HALT
		←	Mute	
HLTB	HLTB	→		HALT
		←	Mute	
ATTRIB	ATTRIB(0,0)	→		HALT
		←	Mute	
Slot-MARKER	SLOTMARKER(n)[a]	→		HALT
		←	Mute	
ISO/IEC 14443-4 command	$I(0)_0$(TEST_COMMAND1(1))	→		HALT
		←	Mute	
DESELECT	S(DESELECT)	→		HALT
		←	Mute	
WUPB[e] (Unmatched AFI)	WUPB(1,nAFI)	→		IDLE
		←	Mute	
REQB[e] (Unmatched AFI)	REQB(1,nAFI)	→		HALT
		←	Mute	
HLTB (Unmatched PUPI)	HLTB(~PUPI)	→		HALT
		←	Mute	
ATTRIB (Unmatched PUPI)	ATTRIB(0, 0, ~PUPI)	→		HALT
		←	Mute	
WUPB[b]	WUPB(16)[c]	→		READY-REQUESTED
		←	Mute	

[a] n shall run through all values 2 ≤ n ≤ 16.

[b] If the PICC does not support the REQB/WUPB with N > 1 (see ISO/IEC 14443-3:2018, 7.6.1), the test need not be applied.

[c] Nevertheless, there is statistically a probability of 1/16 so that the PICC answers ATQB and goes to READY-DECLARED sub-state.

[d] All matched AFIs as defined by the PICC manufacturer shall be tested.

[e] Any unmatched AFI value, also values which are RFU, should be tested.

G.4.4.6.3 Test report

Fill the appropriate row in Table G.113 according to Table G.52.

Table G.52 — Result criteria for Scenario G.26: Behavior of the PICC Type B in the HALT state

Explanation	Test result
Only when the PICC responded as indicated in the procedure	PASS
Any other case	FAIL

G.4.4.7 Scenario G.27: Behavior of the PICC Type B in the PROTOCOL state

G.4.4.7.1 Scope

This test is to determine the behavior of the PICC Type B in the PROTOCOL state according to ISO/IEC 14443-4:2018, 7.4.7. This test shall ensure that the activated PICC does not respond to any initialization command.

G.4.4.7.2 Procedure

Perform the following steps for every row of Table G.53:

a) Put the PICC into PROTOCOL state.

b) Perform the state transition by sending the command as indicated in the PICC-test-apparatus column.

c) Check if the PICC response is as indicated in the PICC column.

d) Check if the PICC is in the Test Target State (TTS).

Table G.53 — Transitions from PROTOCOL state

Transition	PICC-test-apparatus		PICC	TTS
REQB[b]	REQB(1)	→		PROTOCOL
		←	Mute	
WUPB[b]	WUPB(1)	→		PROTOCOL
		←	Mute	
REQB (wrong CRC)	('05 00 00' ~CRC_B)	→		PROTOCOL
		←	Mute	
WUPB (wrong CRC)	('05 00 08' ~CRC_B)	→		PROTOCOL
		←	Mute	
HLTB	HLTB	→		PROTOCOL
		←	Mute	
ATTRIB	ATTRIB(0,0)	→		PROTOCOL
		←	Mute	
Slot-MARKER	SLOTMARKER(n)[a]	→		PROTOCOL
		←	Mute	
ISO/IEC 14443-4 command	$I(0)_0$(TEST_COMMAND1(1))	→		PROTOCOL
		←	$I(0)_0$(TEST_RESPONSE1(1))	
DESELECT	S(DESELECT)	→		HALT
		←	S(DESELECT)	
WUPB[c] (unmatched AFI)	WUPB(1,nAFI)	→		PROTOCOL
		←	Mute	
REQB[c] (unmatched AFI)	REQB(1,nAFI)	→		PROTOCOL
		←	Mute	

[a] n shall run through all values 2 ≤ n ≤ 16.

[b] All matched AFIs as defined by the PICC manufacturer shall be tested.

[c] Any unmatched AFI values, also values which are currently RFU, should be tested.

[d] Check first IDLE state as TTS. If TTS is not IDLE state, rerun the test a second time to check PROTOCOL state.

Table G.53 *(continued)*

Transition	PICC-test-apparatus		PICC	TTS
HLTB (unmatched PUPI)	HLTB(~PUPI)	→		PROTOCOL
		←	Mute	
ATTRIB (unmatched PUPI)	ATTRIB(0, 0, ~PUPI)	→		PROTOCOL
		←	Mute	
Type A command	REQA	→		IDLE or PROTOCOL[d]
		←	Mute	

[a] n shall run through all values $2 \leq n \leq 16$.

[b] All matched AFIs as defined by the PICC manufacturer shall be tested.

[c] Any unmatched AFI values, also values which are currently RFU, should be tested.

[d] Check first IDLE state as TTS. If TTS is not IDLE state, rerun the test a second time to check PROTOCOL state.

G.4.4.7.3 Test report

Fill the appropriate row in Table G.113 according to Table G.54.

Table G.54 — Result criteria for Scenario G.27: Behavior of the PICC Type B in the PROTOCOL state

Explanation	Test result
Only when the PICC responded as indicated in the procedure	PASS
Any other case	FAIL

G.4.5 Scenario G.28: Handling of Type B anticollision

G.4.5.1 Scope

The purpose of this test is to determine the handling of a PICC Type B anticollision according to ISO/IEC 14443-3:2018, 7.4.1.

The core of this test is the procedure AnticollisionB(N, outparam chi2) which is defined in the pseudo code below. The procedure performs 256 REQB(N) commands and following Slot-MARKER commands and counts how many times each of the N slots has been selected by the PICC. The procedure also checks if the PICC has mapped each REQB(N) request to exactly one slot. If this is not the case the test returns FAIL.

Since Type B anticollision is based on random selection of the slots, statistical methods shall be used for verification. As it is the nature of all statistical tests, this test can fail even in the case the PICC behaves correctly. This failure is called a “Type I error” in statistical terms. This error cannot be completely avoided. Instead, the probability of its occurrence can be controlled by the so called “significance value” α. This means, the smaller α, the less probable the “Type I error”. However, this does not mean that one should select α as small as possible. This is because the smaller α is, the more probable is that the test passes a bad PICC (i.e. a PICC that doesn’t select the slots with the right probability). In statistical terms this is called a “Type II error”.

The PICC shall additionally select each of the N slots with equal probability (i.e. 1/N). In order to verify this, the statistical χ^2-test on all slots shall be performed. The result of this test is the value chi2 which shall be compared against the $\chi^2_{\alpha,N-1}$ quintile.

G.4.5.2 Procedure

If one of the statistical tests fails in step e) the test lab may rerun the test for this parameter N.

Perform the following steps for each value N = 2, 4, 8, 16.

a) Set the significance level α to 0,005 and lookup from Table G.55 the corresponding $\chi^2_{\alpha,N-1}$ quintile. Other choices of α according to Table G.51 are optional for the test applicants.

b) Reset the PICC.

c) Execute AnticollisionB(N, chi2).

d) If AnticollisionB returns FAIL, fail the test.

e) If chi2 ≤ $\chi^2_{\alpha,N-1}$ then the test is PASS else the test is FAIL.

Table G.55 — α quintile values

α	$\chi^2_{\alpha,N-1}$			
	$\chi^2_{\alpha,1}$	$\chi^2_{\alpha,3}$	$\chi^2_{\alpha,7}$	$\chi^2_{\alpha,15}$
0,1 (optional)	2,706	6,251	12,017	22,307
0,05 (optional)	3,841	7,815	14,067	24,996
0,01 (optional)	6,635	11,345	18,475	30,578
0,005	7,879	12,838	20,278	32,801

Pseudo code: Type B anticollision procedure

```
1   Procedure AnticollisionB(N, chi2)
3   // TPDUSend and TPDURecv are PCD specific functions
4   // to send and receive TPDU frames
5   //
6   // probability for selecting slot
7   p = 1/N
8   //
9   // clear slot counters
10  for i from 1 to N do
11    Slots[i] = 0
12  endfor
13  //
14  // collect data
15  for i from 1 to 256 do
16    Reset the PICC
17    TPDUSend (REQB(N))
18    if TPDURecv() = ATQB then
19      Slots[1] = Slots[1]+1
20    endif
21    for j from 2 to N do
22      TPDUSend (SLOTMARKER(j))
23      if TPDURecv () = ATQB then
24        Slots[j] = Slots[j]+1
25      endif
26    endfor
27  endfor
28  //
29  // check that exactly
30  // one slot has been selected at each run
31  cnt = 0
32  for i from 1 to N do
33    cnt = cnt + Slots[i]
34  endfor
35 if cnt ≠ 256 then
36    return FAIL
37  endif
38  Chi2 = 0
39  for i from 1 to N do
40    chi2 = chi2 + Slots[i]*Slots[i]
41  endfor
42  chi2 = chi2*N/256 - 256
```

```
43  return PASS
```

NOTE Continue with step e) only if PASS is returned in line 43.

G.4.5.3 Test report

Fill the appropriate row in Table G.113 according to Table G.56.

Table G.56 — Result criteria for Scenario G.28: Handling of Type B anticollision

Explanation	Test result
Only when every Anticollision Test procedure has returned PASS	PASS
When any Anticollision Test procedure has returned the value FAIL	FAIL

G.4.6 Handling of ATTRIB

G.4.6.1 Scope

This test is to determine the behavior of the PICC Type B on ATTRIB command according to ISO/IEC 14443-3:2018, 7.10.

G.4.6.2 Procedure

For each of the scenarios described in Table G.57 and Table G.58, perform the following steps:

a) Put the PICC into READY-DECLARED sub-state.

b) Send the command sequence as described in the PICC-test-apparatus column.

c) Check that the response of the PICC conforms with the one given in the PICC column.

d) Check if the PICC is in PROTOCOL state.

Table G.57 — Scenario G.29: ATTRIB with wrong PUPI

PICC-test-apparatus		PICC
('1D' ~PUPI '00 00 01 00' CRC_B)	→	
	←	Mute
ATTRIB(0,0)	→	
	←	Answer to ATTRIB(0)

Table G.58 — Scenario G.30: ATTRIB after wrong ATTRIB

PICC-test-apparatus		PICC
('1D' PUPI '00 00 01 00' ~CRC_B)	→	
	←	Mute
ATTRIB(0,0)	→	
	←	Answer to ATTRIB(0)

G.4.6.3 Test report

Fill the appropriate row in Table G.113 according to Table G.59.

Table G.59 — Result criteria for handling of ATTRIB

Explanation	Test result
Only when the PICC responded as indicated in the procedure	PASS
Any other case	FAIL

G.4.7 Scenario G.31: Handling of Maximum Frame Size

G.4.7.1 Scope

This test is to determine if the PICC Type B respects the Maximum Frame Size as negotiated by the ATTRIB according to ISO/IEC 14443-3:2018, 7.10.4.

G.4.7.2 Procedure

Perform the following steps for each Maximum Frame Size Code in ATTRIB as defined in ISO/IEC 14443-3:

a) Put the PICC into READY-DECLARED sub-state as described in G.4.4.2.2.

b) Send the ATTRIB(0, fsdi) command with parameter fsdi as in the particular test.

c) Check if the PICC answer is Answer to ATTRIB(0).

NOTE The PICC may require additional sequences to be ready to accept TEST_COMMAND2(2).

d) Send the I-block $I(0)_0$(TEST_COMMAND2(2)).

e) Check if the size of the I-block response of the PICC response is ≤ Maximum Frame Size.

G.4.7.3 Test report

Fill the appropriate row in Table G.113 according to Table G.60.

Table G.60 — Result criteria for Scenario G.31: Handling of Maximum Frame Size

Explanation	Test result
Only when the PICC responded as indicated in the procedure	PASS
Any other case	FAIL

G.4.8 Handling of TR2 and SFGT

G.4.8.1 Scope

This test is to determine if the PICC Type B respects the minimum TR2 according ISO/IEC 14443-3:2018, 7.9.4.4 and SFGT according ISO/IEC 14443-3:2018, 7.9.4.7.

G.4.8.2 Procedure

This procedure shall be repeated for two combinations of bit rates as described below:

1) at a bit rate of $f_c/128$ in both directions;

2) at the maximum bit rates supported by the PICC in each direction.

Perform the following steps of the scenario described in Table G.61:

a) Put the PICC into IDLE state.

b) For each step in the scenario do:

1) Send the command as described in the PICC-test-apparatus column respecting the timing condition described in the TR2 column.

2) Check that the PICC response matches the one of the PICC column.

c) End for.

Table G.61 — Scenario G.73: Handling of TR2 and SFGT

Step	TR2 to apply	PICC-test-apparatus		PICC
1	—	REQB(1)[a]	→	
			←	ATQB or Extended ATQB
2	Minimum TR2 + $50/f_c$[b,c]	ATTRIB(0,0)	→	
			←	Answer to ATTRIB
3	Minimum TR2 + $50/f_c$[b,c] or SFGT + $50/f_c$[c,d]	$I(0)_0$(TEST_COMMAND1(1))	→	
			←	$I(0)_0$(TEST_RESPONSE1(1))[e]
4	Minimum TR2 + $50/f_c$[b,c]	$I(0)_1$(TEST_COMMAND3)	→	
			←	S(WTX)(WTXM)
5	Minimum TR2 + $50/f_c$[b,c]	S(WTX)(WTXM)	→	
			←	$I(0)_1$(TEST_RESPONSE3)[f]

[a] Bit 5 of PARAM is set to 1 (PICC-test-apparatus supports the extended ATQB).

[b] Minimum TR2 is set as given in the ATQB.

[c] The applied TR2 or SFGT shall have a maximum tolerance of $\pm 25/f_c$.

[d] If the PICC has responded with extended ATQB.

[e] S(WTX) request(s) may be sent by the PICC. The PICC-test-apparatus shall acknowledge each S(WTX) request with an S(WTX) response until receiving the PICC response. There is no specific requirement on TR2 for the S(WTX) response(s).

[f] Additional S(WTX) request(s) may be sent by the PICC. The PICC-test-apparatus shall acknowledge each S(WTX) request with an S(WTX) response until receiving the PICC response. There is no specific requirement on TR2 for the S(WTX) response(s).

G.4.8.3 Test report

Fill the appropriate row in Table G.113 according to Table G.62.

Table G.62 — Result criteria for handling of TR2

Explanation	Test result
Only when the PICC responded as indicated in the procedure	PASS
Any other case	FAIL

G.5 Test methods for logical operation of the PICC Type A or Type B

G.5.1 General

G.5.1.1 General

This subclause contains tests verifying that the activated PICC conforms to the ISO/IEC 14443-4. This subclause applies to PICC Type A and Type B.

G.5.1.2 PICC activation process

G.5.1.2.1 General

PICC activation is the process of putting the PICC in the state where protocol blocks defined in ISO/IEC 14443-4 may be exchanged. This process is dependent on the PICC communication signal interface.

NOTE The PICC may require additional sequences to be ready to accept step 1 of the scenario.

G.5.1.2.2 Activation of the PICC Type A

a) Put the PICC into ACTIVE state as described in G.3.3.2.2.

b) Send RATS(cid, fsdi).

c) Check that the PICC response is a valid ATS.

G.5.1.2.3 Activation of the PICC Type B

a) Put the PICC into READY-DECLARED sub-state as described in G.4.4.2.2.

b) Send ATTRIB(cid, fsdi).

c) Check that the PICC response is a valid Answer to ATTRIB (cid).

G.5.2 PICC reaction to ISO/IEC 14443-4 scenarios

G.5.2.1 Scope

This test is to determine the behavior of the PICC according to ISO/IEC 14443-4:2018, Clause 7. This test uses implementations of the protocol scenarios of ISO/IEC 14443-4:2018, Annex B.

G.5.2.2 Procedure

For each of the scenarios described in Table G.63 to Table G.86, perform the following steps:

a) Activate the PICC as described in G.5.1.2, use CID = 0.

b) For each step in the scenario do:

   1) Send the command as described in the PICC-test-apparatus column.

   2) Check that the PICC response matches the one of the PICC column.

c) End for.

Table G.63 — Scenario G.32: Exchange of I-blocks

Step	PICC-test-apparatus		PICC
1	$I(0)_0$(TEST_COMMAND1(1))	→	
		←	$I(0)_0$(TEST_RESPONSE1(1))
2	$I(0)_1$(TEST_COMMAND1(1))	→	
		←	$I(0)_1$(TEST_RESPONSE1(1))

Table G.64 — Scenario G.33: Request for waiting time extension

Step	PICC-test-apparatus		PICC
1	$I(0)_0$(TEST_COMMAND3)	→	
		←	S(WTX)(WTXM)
2	S(WTX)(WTXM)	→	
		←	$I(0)_0$(TEST_RESPONSE3)
3	$I(0)_1$(TEST_COMMAND1(1))	→	
		←	$I(0)_1$(TEST_RESPONSE1(1))

Table G.65 — Scenario G.34: DESELECT

Step	PICC-test-apparatus		PICC
1	$I(0)_0$(TEST_COMMAND1(1))	→	
		←	$I(0)_0$(TEST_RESPONSE1(1))
2	S(DESELECT)	→	
		←	S(DESELECT)
3	REQA or REQB(1)[a]	→	
		←	Mute
4	WUPA or WUPB(1)[a]	→	
		←	ATQA or ATQB[a]

[a] For the PICC Type A, the left option shall be used. For the PICC Type B, the right option shall be used.

Table G.66 — Scenario G.35: PCD uses chaining

Step	PICC-test-apparatus		PICC
1	$I(1)_0$(TEST_COMMAND1$(2)_1$)	→	
		←	$R(ACK)_0$
2	$I(0)_1$(TEST_COMMAND1$(2)_2$)	→	
		←	$I(0)_1$(TEST_RESPONSE1(2))
3	$I(0)_0$(TEST_COMMAND1(1))	→	
		←	$I(0)_0$(TEST_RESPONSE1(1))

Table G.67 — Scenario G.36: PICC uses chaining

Step	PICC-test-apparatus		PICC
1	$I(0)_0$(TEST_COMMAND2(2))	→	
		←	$I(1)_0$(TEST_RESPONSE2$(2)_1$)
2	$R(ACK)_1$	→	
		←	$I(0)_1$(TEST_RESPONSE2$(2)_2$)
3	$I(0)_0$(TEST_COMMAND1(1))	→	
		←	$I(0)_0$(TEST_RESPONSE1(1))

Table G.68 — Scenario G.37: Start of protocol

Step	PICC-test-apparatus		PICC
1	$I(0)_0$(TEST_COMMAND1(1), ~CRC)	→	
		←	Mute

Table G.68 *(continued)*

Step	PICC-test-apparatus		PICC
2	$R(NAK)_0$	→	
		←	$R(ACK)_1$
3	$I(0)_0$(TEST_COMMAND1(1))	→	
		←	$I(0)_0$(TEST_RESPONSE1(1))
4	$I(0)_1$(TEST_COMMAND1(1))	→	$I(0)_1$(TEST_RESPONSE1(1))
		←	

Table G.69 — Scenario G.38: Exchange of I-blocks

Step	PICC-test-apparatus		PICC
1	$I(0)_0$(TEST_COMMAND1(1))	→	
		←	$I(0)_0$(TEST_RESPONSE1(1))
2	$I(0)_1$(TEST_COMMAND1(1), ~CRC)	→	
		←	Mute
3	$R(NAK)_1$	→	
		←	$R(ACK)_0$
4	$I(0)_1$(TEST_COMMAND1(1))	→	
		←	$I(0)_1$(TEST_RESPONSE1(1))
5	$I(0)_0$(TEST_COMMAND1(1))	→	
		←	$I(0)_0$(TEST_RESPONSE1(1))

Table G.70 — Scenario G.39: Exchange of I-blocks 1

Step	PICC-test-apparatus		PICC
1	$I(0)_0$(TEST_COMMAND1(1))	→	
		←	$I(0)_0$(TEST_RESPONSE1(1))
2	$R(NAK)_0$	→	
		←	$I(0)_0$(TEST_RESPONSE1(1))
3	$I(0)_1$(TEST_COMMAND1(1))	→	
		←	$I(0)_1$(TEST_RESPONSE1(1))

Table G.71 — Scenario G.40: Exchange of I-blocks 2

Step	PICC-test-apparatus		PICC
1	$I(0)_0$(TEST_COMMAND1(1))	→	
		←	$I(0)_0$(TEST_RESPONSE1(1))
2	$R(NAK, \sim CRC)_0$	→	
		←	Mute
3	$R(NAK)_0$	→	
		←	$I(0)_0$(TEST_RESPONSE1(1))
4	$I(0)_1$(TEST_COMMAND1(1))	→	
		←	$I(0)_1$(TEST_RESPONSE1(1))

Table G.72 — Scenario G.41: Request for waiting time extension

Step	PICC-test-apparatus		PICC
1	$I(0)_0$(TEST_COMMAND3)	→	
		←	S(WTX)(WTXM)
2	$R(NAK)_0$	→	
		←	S(WTX)(WTXM)
3	S(WTX)(WTXM)	→	
		←	$I(0)_0$(TEST_RESPONSE3)
4	$I(0)_1$(TEST_COMMAND1(1))	→	
		←	$I(0)_1$(TEST_RESPONSE1(1))

Table G.73 — Scenario G.42: Request for waiting time extension

Step	PICC-test-apparatus		PICC
1	$I(0)_0$(TEST_COMMAND3)	→	
		←	S(WTX)(WTXM)
2	$R(NAK, \sim CRC)_0$	→	
		←	Mute
3	$R(NAK)_0$	→	
		←	S(WTX)(WTXM)
4	S(WTX)(WTXM)	→	
		←	$I(0)_0$(TEST_RESPONSE3)
5	$I(0)_1$(TEST_COMMAND1(1))	→	
		←	$I(0)_1$(TEST_RESPONSE1(1))

Table G.74 — Scenario G.43: Request for waiting time extension

Step	PICC-test-apparatus		PICC
1	$I(0)_0$(TEST_COMMAND3)	→	
		←	S(WTX)(WTXM)
2	S(WTX)(WTXM, ~CRC)	→	
		←	Mute
3	$R(NAK)_0$	→	
		←	S(WTX)(WTXM)
4	S(WTX)(WTXM)	→	
		←	$I(0)_0$(TEST_RESPONSE3)
5	$I(0)_1$(TEST_COMMAND1(1))	→	
		←	$I(0)_1$(TEST_RESPONSE1(1))

Table G.75 — Scenario G.74: Request for waiting time extension

Step	PICC-test-apparatus		PICC
1	$I(0)_0$(TEST_COMMAND3)	→	
		←	S(WTX)(WTXM)

[a] The parity error is simulated on the first transmitted byte of the frame by reversing the parity bit

Table G.75 *(continued)*

Step	PICC-test-apparatus		PICC
2	S(WTX) (WTXM, wrong parity bit)[a]	→	
		←	Mute
3	$R(NAK)_0$	→	
		←	S(WTX)(WTXM)
4	S(WTX)(WTXM)	→	
		←	$I(0)_0$(TEST_RESPONSE3)
5	$I(0)_1$(TEST_COMMAND1(1))	→	
		←	$I(0)_1$(TEST_RESPONSE1(1))
[a] The parity error is simulated on the first transmitted byte of the frame by reversing the parity bit			

Table G.76 — Scenario G.44: Request for waiting time extension

Step	PICC-test-apparatus		PICC
1	$I(0)_0$(TEST_COMMAND3)	→	
		←	S(WTX)(WTXM)
2	S(WTX)(WTXM)	→	
		←	$I(0)_0$(TEST_RESPONSE3)
3	$R(NAK)_0$	→	
		←	$I(0)_0$(TEST_RESPONSE3)
4	$I(0)_1$(TEST_COMMAND1(1))	→	
		←	$I(0)_1$(TEST_RESPONSE1(1))

Table G.77 — Scenario G.45: Request for waiting time extension

Step	PICC-test-apparatus		PICC
1	$I(0)_0$(TEST_COMMAND3)	→	
		←	S(WTX)(WTXM)
2	S(WTX)(WTXM)	→	
		←	$I(0)_0$ (TEST_RESPONSE3)
3	$R(NAK, \sim CRC)_0$	→	
		←	Mute
4	$R(NAK)_0$	→	
		←	$I(0)_0$ (TEST_RESPONSE3)
5	$I(0)_1$(TEST_COMMAND1(1))	→	
		←	$I(0)_1$(TEST_RESPONSE1(1))

Table G.78 — Scenario G.46: DESELECT

Step	PICC-test-apparatus		PICC
1	$I(0)_0$(TEST_COMMAND1(1))	→	
		←	$I(0)_0$(TEST_RESPONSE1(1))
2	S(DESELECT, ~CRC)	→	
		←	Mute
[a] For the PICC Type A, the left option shall be used. For the PICC Type B, the right option shall be used.			

Table G.78 *(continued)*

Step	PICC-test-apparatus		PICC
3	S(DESELECT)	→	
		←	S(DESELECT)
4	REQA or REQB(1)[a]	→	
		←	Mute
5	WUPA or WUPB(1)[a]	→	
		←	ATQA or ATQB[a]

[a] For the PICC Type A, the left option shall be used. For the PICC Type B, the right option shall be used.

Table G.79 — Scenario G.47: PCD uses chaining

Step	PICC-test-apparatus		PICC
1	$I(1)_0$(TEST_COMMAND1(3)$_1$)	→	
		←	$R(ACK)_0$
2	$R(NAK)_0$	→	
		←	$R(ACK)_0$
3	$I(1)_1$(TEST_COMMAND1(3)$_2$)	→	
		←	$R(ACK)_1$
4	$I(0)_0$(TEST_COMMAND1(3)$_3$)	→	
		←	$I(0)_0$(TEST_RESPONSE1(3))
5	$I(0)_1$(TEST_COMMAND1(1))	→	
		←	$I(0)_1$(TEST_RESPONSE1(1))

Table G.80 — Scenario G.48: PCD uses chaining

Step	PICC-test-apparatus		PICC
1	$I(1)_0$(TEST_COMMAND1(3)$_1$)	→	
		←	$R(ACK)_0$
2	$I(1)_1$(TEST_COMMAND1(3)$_2$, ~CRC)	→	
		←	Mute
3	$R(NAK)_1$	→	
		←	$R(ACK)_0$
4	$I(1)_1$(TEST_COMMAND1(3)$_2$)	→	
		←	$R(ACK)_1$
5	$I(0)_0$(TEST_COMMAND1(3)$_3$)	→	
		←	$I(0)_0$(TEST_RESPONSE1(3))
6	$I(0)_1$(TEST_COMMAND1(1))	→	
		←	$I(0)_1$(TEST_RESPONSE1(1))

Table G.81 — Scenario G.49: PCD uses chaining

Step	PICC-test-apparatus		PICC
1	$I(1)_0$ (TEST_COMMAND1(3)$_1$)	→	
		←	$R(ACK)_0$
2	$R(NAK, \sim CRC)_0$	→	
		←	Mute

Table G.81 *(continued)*

Step	PICC-test-apparatus		PICC
3	$R(NAK)_0$	→	
		←	$R(ACK)_0$
4	$I(1)_1(TEST\_COMMAND1(3)_2)$	→	
		←	$R(ACK)_1$
5	$I(0)_0(TEST\_COMMAND1(3)_3)$	→	
		←	$I(0)_0$ (TEST_RESPONSE1(3))
6	$I(0)_1(TEST\_COMMAND1(1))$	→	
		←	$I(0)_1(TEST\_RESPONSE1(1))$

Table G.82 — Scenario G.50: PICC uses chaining

Step	PICC-test-apparatus		PICC
1	$I(0)_0(TEST\_COMMAND2(3))$	→	
		←	$I(1)_0(TEST\_RESPONSE2(3)_1)$
2	$R(ACK, \sim CRC)_1$	→	
		←	Mute
3	$R(ACK)_1$	→	
		←	$I(1)_1(TEST\_RESPONSE2(3)_2)$
4	$R(ACK)_0$	→	
		←	$I(0)_0(TEST\_RESPONSE2(3)_3)$
5	$I(0)_1(TEST\_COMMAND1(1))$	→	
		←	$I(0)_1(TEST\_RESPONSE1(1))$

Table G.83 — Scenario G.51: PICC uses chaining

Step	PICC-test-apparatus		PICC
1	$I(0)_0(TEST\_COMMAND2(3))$	→	
		←	$I(1)_0(TEST\_RESPONSE2(3)_1)$
2	$R(ACK)_1$	→	
		←	$I(1)_1(TEST\_RESPONSE2(3)_2)$
3	$R(ACK)_1$	→	
		←	$I(1)_1(TEST\_RESPONSE2(3)_2)$
4	$R(ACK)_0$	→	
		←	$I(0)_0(TEST\_RESPONSE2(3)_3)$
5	$I(0)_1(TEST\_COMMAND1(1))$	→	
		←	$I(0)_1(TEST\_RESPONSE1(1))$

Table G.84 — Scenario G.52: PICC uses chaining

Step	PICC-test-apparatus		PICC
1	$I(0)_0(TEST\_COMMAND2(2))$	→	
		←	$I(1)_0(TEST\_RESPONSE2(2)_1)$
2	$R(NAK)_0$	→	
		←	$I(1)_0(TEST\_RESPONSE2(2)_1)$

Table G.84 *(continued)*

Step	PICC-test-apparatus		PICC
3	$R(ACK)_1$	→	
		←	$I(0)_1(TEST\_RESPONSE2(2)_2)$
4	$I(0)_0(TEST\_COMMAND1(1))$	→	
		←	$I(0)_0(TEST\_RESPONSE1(1))$

Table G.85 — Scenario G.53: PICC Presence Check Method 1

Step	PICC-test-apparatus		PICC
1	$I(empty)_0$	→	
		←	$I()_0$
2	$I(0)_1(TEST\_COMMAND1(1))$	→	
		←	$I(0)_1(TEST\_RESPONSE1(1))$
3	$I(empty)_0$	→	
		←	$I()_0$

Table G.86 — Scenario G.54: PICC Presence Check Method 2

Step	PICC-test-apparatus		PICC
1	$R(NAK)_0$	→	
		←	$R(ACK)_1$
2	$R(NAK)_0$	→	
		←	$R(ACK)_1$
3	$I(0)_0(TEST\_COMMAND1(1))$	→	
		←	$I(0)_0(TEST\_RESPONSE1(1))$
4	$R(NAK)_1$	→	
		←	R(ACK)0
5	$I(0)_1(TEST\_COMMAND1(1))$	→	
		←	$I(0)_1(TEST\_RESPONSE1(1))$

G.5.2.3 Test report

Fill the appropriate rows in Table G.114 according to Table G.87.

Table G.87 — Result criteria for PICC reaction to ISO/IEC 14443-4 scenarios

Explanation	Test result
Only when the PICC responded as indicated in the procedure	PASS
Any other case	FAIL

G.5.3 Handling of PICC error detection

G.5.3.1 Scope

This test is to determine the error detection mechanism of the PICC as described in ISO/IEC 14443-4:2018, 7.5.7.

G.5.3.2 Procedure

For each of the scenarios described in Table G.88, Table G.89 and Table G.90, perform the following steps:

a) Place the PICC into the field.

b) Activate the PICC as described in G.5.1.2, use CID = 0.

c) For each step in scenario do:

   1) Send the command as described in the PICC-test-apparatus column.

   2) Check if the PICC response is as described in the PICC column.

d) End for.

Table G.88 — Scenario G.55: Wrong CRC on I-block

Step	PICC-test-apparatus		PICC	Comment
1	$I(0)_0$(TEST_COMMAND1(1), ~CRC)	→		ISO/IEC 14443-4: 2018, 7.5.7.1 a)
		←	Mute	
2	$I(0)_0$(TEST_COMMAND1(1))	→		
		←	$I(0)_0$(TEST_RESPONSE1(1))	

Table G.89 — Scenario G.56: Wrong CRC on chained I-block

Step	PICC-test-apparatus		PICC	Comment
1	$I(1)_0$(TEST_COMMAND1(2)$_1$)	→		
		←	$R(ACK)_0$	
2	$I(0)_1$(TEST_COMMAND1(2)$_2$, ~CRC)	→		ISO/IEC 14443-4: 2018, 7.5.7.1 a)
		←	Mute	
3	$I(0)_1$(TEST_COMMAND1(2)$_2$)	→		
		←	$I(0)_1$(TEST_RESPONSE1(2))	

Table G.90 — Scenario G.57: Wrong CRC on S(WTX)-block

Step	PICC-test-apparatus		PICC	Comment
1	$I(0)_0$(TEST_COMMAND3)	→		
		←	S(WTX)(WTXM)	
2	S(WTX)(WTXM, ~CRC)	→		ISO/IEC 14443-4: 2018, 7.5.7.1 a)
		←	Mute	
3	S(WTX)(WTXM)	→		
		←	$I(0)_0$(TEST_RESPONSE3)	

G.5.3.3 Test report

Fill the appropriate rows in Table G.114 according to Table G.91.

Table G.91 — Result criteria for handling of PICC error detection

Explanation	Test result
Only when the PICC responded as indicated in the procedure	PASS
Any other case	FAIL

G.5.4 PICC reaction on CID

G.5.4.1 Scope

This test is to determine the reaction of the PICC to CID coding according to ISO/IEC 14443-4:2018, 7.1.2.2.

G.5.4.2 Procedure

For each of the scenarios described in Table G.94 to Table G.99, perform the following steps. Use the proper CID test case table depending upon whether the PICC supports CID or not.

For each row in the CID test case tables Table G.92 or Table G.93 do:

a) Activate the PICC with cid_{ass} as indicated in the column Assigned CID.

b) Perform a block exchange as described in the corresponding scenario. Use the cid_{cmd} as described in the Command CID column in the CID test case table.

c) Check if the PICC response matches with the response as in the PICC column in the scenario. If two response options are indicated for the PICC, then the unique expected response will be determined from the expected PICC response column in the CID test case table.

Table G.92 — CID test cases (for PICCs which support CID)

Test No.[a]	Assigned CID (cid_{ass})	Command CID (cid_{cmd})	Expected PICC response
1	1	1	Response 1 of the scenario
2	0	0	Response 1 of the scenario
3	0	NO CID	Response 1 of the scenario
4	1	NO CID	Response 2 of the scenario (Mute)
5	0	1	Response 2 of the scenario (Mute)
6	1	0	Response 2 of the scenario (Mute)
7	2	1	Response 2 of the scenario (Mute)

[a] Each test shall be carried out with each of the scenarios described.

Table G.93 — CID test cases (for PICCs which do not support CID)

Test No.[a]	Assigned CID (cid_{ass})	Command CID (cid_{cmd})	Expected PICC response
1	0	0	Response 2 of the scenario (Mute)
2	0	NO CID	Response 1 of the scenario
3	0	1	Response 2 of the scenario (Mute)
4[b]	1	NO CID	Response 1 of the scenario

[a] Each test shall be carried out with each of the scenarios described.

[b] Applies to Type A PICC only.

Table G.94 — Scenario G.58: CID on I-block

Step	PICC-test-apparatus		PICC
1	$I(0)_0$(TEST_COMMAND1(1), CID = cid_{cmd})	→	
		←	Response 1: $I(0)_0$(TEST_RESPONSE1(1), CID = cid_{cmd}) Response 2: Mute[a]

[a] Response 1 or response 2 according to Table G.58 or Table G.59.

Table G.95 — Scenario G.59: CID on I-block with chaining

Step	PICC-test-apparatus		PICC
1	$I(1)_0$ (TEST_COMMAND1$(2)_1$,CID = cid_{cmd})	→	
		←	Response 1: R(ACK, CID = $cid_{cmd})_0$ Response 2: Mute[a]
[a] Response 1 or response 2 according to Table G.58 or Table G.59.			

Table G.96 — Scenario G.60: CID on R-block

Step	PICC-test-apparatus		PICC
1	$I(0)_0$(TEST_COMMAND2(3), CID[a] = cid_{ass})	→	
		←	$I(1)_0$(TEST_RESPONSE2$(3)_1$, CID = cid_{ass})
2	R(ACK, CID[a] = $cid_{cmd})_1$	→	
		←	Response 1: $I(1)_1$ (TEST_RESPONSE2$(3)_2$, CID = cid_{cmd}) Response 2: Mute[b]
[a] For PICC not supporting CID, use no CID.			
[b] Response 1 or response 2 according to Table G.58 or Table G.59.			

Test No. 3 of Table G.92 shall be omitted for this scenario.

Table G.97 — Scenario G.61: CID on S(WTX)-block

Step	PICC-test-apparatus		PICC
1	$I(0)_0$(TEST_COMMAND3, CID[a] = cid_{ass})	→	
		←	S(WTX)(WTXM, CID = cid_{ass})
2	S(WTX)(WTXM, CID[a] = cid_{cmd})	→	
		←	Response 1: $I(0)_0$(TEST_RESPONSE3, CID[a] = cid_{cmd}) Response 2: Mute[b]
[a] For PICC not supporting CID, use no CID.			
[b] Response 1 or response 2 according to Table G.58 or Table G.59.			

Test No. 3 of Table G.92 shall be omitted for this scenario

Table G.98 — Scenario G.62: CID on S(DESELECT)-block

Step	PICC-test-apparatus		PICC
1	S(DESELECT, CID = cid_{cmd})	→	
		←	Response 1: S(DESELECT, CID = cid_{cmd}) Response 2: Mute[a]
[a] Response 1 or response 2 according to Table G.58 or Table G.59.			

Table G.99 — Scenario G.66: CID on S(PARAMETERS)-block

Step	PICC-test-apparatus		PICC
1	S(PARAMETERS, CID = cid_{cmd})	→	
		←	Response 1: S(PARAMETERS, CID = cid_{cmd}) Response 2: Mute[a]

[a] If the PICC supports S(PARAMETERS) response 1 or response 2 in accordance with Table G.58 or Table G.59. Else always response 2.

G.5.4.3 Test report

Fill the appropriate row in Table G.114 according to Table G.100.

Table G.100 — Result criteria for PICC reaction on CID

Explanation	Test result
Only when the PICC responded as indicated in the procedure	PASS
Any other case	FAIL

G.5.5 PICC reaction on NAD

G.5.5.1 Scope

This test is to determine the reaction of the PICC to NAD coding according to ISO/IEC 14443-4:2018, 7.1.2.3.

G.5.5.2 Procedure

For each of the scenarios described in Table G.101, Table G.102 and Table G.103, perform the following steps:

Activate the PICC as described in clause PICC activation process G.5.1.2, use CID = 0 and FSDI = 0,

For each step in the scenario do:

a) Send the command as described in the PICC-test-apparatus column.

b) Check that the PICC response matches the one of the PICC column.

Let n be an arbitrary value of a valid NAD with b4 and b8 set to (0)b.

Table G.101 — Scenario G.63: NAD on I-block (for PICCs supporting NAD)

Step	PICC-test-apparatus		PICC
1	$I(0)_0$(TEST_COMMAND1(1), NAD = n)	→	
		←	$I(0)_0$(TEST_RESPONSE1(1), containing NAD)

Table G.102 — Scenario G.64: NAD on chained I-block (for PICCs supporting NAD)

Step	PICC-test-apparatus		PICC
1	$I(0)_0$(TEST_COMMAND2(3), NAD = n)	→	
		←	$I(1)_0$(TEST_RESPONSE2$(3)_1$, containing NAD)

Table G.102 *(continued)*

Step	PICC-test-apparatus		PICC
2	$R(ACK)_1$	→	
		←	$I(1)_1$(TEST_RESPONSE2$(3)_2$, not containing NAD)

Table G.103 — Scenario G.65: NAD on I-block (for PICCs not supporting NAD)

Step	PICC-test-apparatus		PICC
1	$I(0)_0$ (TEST_COMMAND1(1), NAD = n)	→	
		←	Mute

G.5.5.3 Test report

Fill the appropriate rows in Table G.114 according to Table G.104.

Table G.104 — Result criteria for PICC reaction on NAD

Explanation	Test result
If the scenario is not applicable for the PICC	Not applicable (N/A)
If the scenario is applicable for the PICC and only when the PICC's response is as indicated in the procedure	PASS
Any other case	FAIL

G.5.6 PICC reaction on S(PARAMETERS) blocks

G.5.6.1 Scope

This test is to determine the behavior of the PICC according to ISO/IEC 14443-4:2018, 7.5.1. This test uses implementations of the protocol scenarios of ISO/IEC 14443-4:2018, B.2.6.

G.5.6.2 Procedure

For each of the scenarios described in Table G.105 to Table G.108, perform the following steps:

a) Activate the PICC as described in G.5.1.2, use CID = 0.

b) For each step in the scenario do:

   1) Send the command as described in the PICC-test-apparatus column.

   2) Check that the PICC response matches the one of the PICC column. If the PICC does not support S(PARAMETERS), the PICC shall stay Mute on every S(PARAMETER) request.

c) End for.

Table G.105 — Scenario G.67: Exchange of additional parameters

Step	PICC-test-apparatus		PICC
1	$I(0)_0$(TEST_COMMAND1(1))	→	
		←	$I(0)_0$(TEST_RESPONSE1(1))
2	S(PARAMETERS)	→	
		←	S(PARAMETERS)[a]

[a] If the PICC does not support S(PARAMETERS), it shall stay Mute.

Table G.105 *(continued)*

Step	PICC-test-apparatus		PICC
3	$I(0)_1$(TEST_COMMAND1(1))	→	
		←	$I(0)_1$(TEST_RESPONSE1(1))
[a] If the PICC does not support S(PARAMETERS), it shall stay Mute.			

Table G.106 — Scenario G.68: Exchange of additional parameters

Step	PICC-test-apparatus		PICC
1	S(PARAMETERS)	→	
		←	S(PARAMETERS)[a]
2	$I(0)_0$(TEST_COMMAND1(1))	→	
		←	$I(0)_0$(TEST_RESPONSE1(1))
[a] If the PICC does not support S(PARAMETERS), it shall stay Mute.			

Table G.107 — Scenario G.69: Exchange of additional parameters

Step	PICC-test-apparatus		PICC
1	S(PARAMETERS)	→	
		←	S(PARAMETERS)[a]
2	S(PARAMETERS)	→	
		←	S(PARAMETERS)[a]
3	$I(0)_0$(TEST_COMMAND1(1))	→	
		←	$I(0)_0$(TEST_RESPONSE1(1))
[a] If the PICC does not support S(PARAMETERS), it shall stay Mute.			

Table G.108 — Scenario G.70: Exchange of additional parameters

Step	PICC-test-apparatus		PICC
1	S(PARAMETERS, ~CRC)	→	
		←	Mute
2	S(PARAMETERS)	→	
		←	S(PARAMETERS)[a]
3	$I(0)_0$(TEST_COMMAND1(1))	→	
		←	$I(0)_0$(TEST_RESPONSE1(1))
[a] If the PICC does not support S(PARAMETERS), it shall stay Mute.			

G.5.6.3 Test report

Fill the appropriate row in Table G.114 according to Table G.109.

Table G.109 — Result criteria for exchange of additional parameters

Explanation	Test result
Only when the PICC responded as indicated in the procedure	PASS
Any other case	FAIL

G.5.7 PICC supporting Type A and Type B

G.5.7.1 Purpose

This test checks that, if the PICC supports Type A and Type B, then it stays locked in the communication signal interface of the first processed Request command until POWER-OFF state (after Answer to Request of one communication signal interface, the other communication signal interface is disabled until the PICC enters POWER-OFF state).

G.5.7.2 Conditions

The PICC-test-apparatus shall be used.

G.5.7.3 Procedure

Perform the following steps:

a) Place the PICC into the test position of the PICC-test-apparatus.

b) Switch the PICC-test-apparatus RF operating field on and wait at least 5 ms.

c) Send a WUPA command and check there is a valid PICC response.

d) Keep the PICC-test-apparatus RF field on for more than 1 s.

e) Send a sequence of 10 WUPB commands and check there is no PICC response.

f) Switch the RF operating field off for a minimum time for resetting a PICC (see ISO/IEC 14443-3:2018, 5.2.4)

g) Switch the PICC-test-apparatus RF operating field on and wait at least 5 ms.

h) Send a WUPB command and check there is a valid PICC response.

i) Keep the PICC-test-apparatus RF field on for more than 1 s.

j) Send a sequence of 10 WUPA commands and check there is no PICC response.

G.5.7.4 Test report

The test result is PASS if all steps of the procedure succeed, otherwise the test result is FAIL.

G.6 Reported results

The results of the tests shall be reported in Table G.110 to Table G.113.

Table G.110 — Type A specific timing table

No	Parameter	Reference	Measured values [$1/f_c$]		Test result PASS or FAIL or N/A
			Min	Max	
1	Frame Delay Time PCD to PICC (for REQA, WUPA, ANTICOLLISION and SELECT commands)	ISO/IEC 14443-3:2018, 6.2.1.1			

[a] PICC response shall comply with ISO/IEC 14443-3:2018, 6.2.1.1. The integer value n corresponding to the measured value shall be additionally indicated.

[b] Start of communication concerns only PICC to PCD bit rates higher than $f_c/128$.

Table G.110 *(continued)*

No	Parameter		Reference	Measured values [$1/f_c$]		Test result PASS or FAIL or N/A
				Min	Max	
2	Activation frame waiting time (for RATS and PPS commands)		ISO/IEC 14443-4:2018, 5.5 ISO/IEC 14443-3:2018, 6.2.1.1			
3	Frame waiting time (for S(DESELECT) and S(PARAMETERS) blocks)		ISO/IEC 14443-4:2018, 7.2 ISO/IEC 14443-4:2018, 8.1 ISO/IEC 14443-3:2018, 6.2.1.1			
4	Frame Delay Time PCD to PICC (for frames other than in previous rows)		ISO/IEC 14443-4:2018, 7.2 ISO/IEC 14443-3:2018, 6.2.1.1	—	—	—
	PCD to PICC bit rate	**PICC to PCD bit rate**		—	—	—
	$f_c/128$	$f_c/128$		a	a	
	$f_c/64$			a	a	
	$f_c/32$			a	a	
	$f_c/16$			a	a	
	$f_c/128$ or $f_c/64$ or $f_c/32$ or $f_c/16$ or $f_c/8$ or $f_c/4$ or $f_c/2$ or $3f_c/4$ or $3f_c/2$ or $2f_c$	$f_c/64$ or $f_c/32$ or $f_c/16$ or $f_c/8$ or $f_c/4$ or $f_c/2$				
5	Start of communication[b]		ISO/IEC 14443-2:2020, 8.2.6.2			

[a] PICC response shall comply with ISO/IEC 14443-3:2018, 6.2.1.1. The integer value n corresponding to the measured value shall be additionally indicated.

[b] Start of communication concerns only PICC to PCD bit rates higher than $f_c/128$.

Table G.111 — Type B specific timing table

No	Parameter	Reference	Measured values [etu] or [$1/f_s$] or [$1/f_c$] depending on the requirements		Test result PASS or FAIL or N/A
			Min	Max	
1	SOF low	ISO/IEC 14443-3:2018, 7.1.4			
2	SOF high	ISO/IEC 14443-3:2018, 7.1.4			
3	EOF low	ISO/IEC 14443-3:2018, 7.1.5			
4	EGT PICC to PCD	ISO/IEC 14443-3:2018, 7.1.2			
5	TR0 for ATQB	ISO/IEC 14443-3:2018, 7.1.6, ISO/IEC 14443-3:2018, 7.10.3			
6	TR1 for ATQB	ISO/IEC 14443-3:2018, 7.1.6, ISO/IEC 14443-3:2018, 7.10.3			
7	TR0 except for ATQB and S(DESELECT) response	ISO/IEC 14443-3:2018, 7.1.6, ISO/IEC 14443-3:2018, 7.10.3			
8	TR1 not for ATQB	ISO/IEC 14443-3:2018, 7.1.6, ISO/IEC 14443-3:2018, 7.10.3			
9	f_s to OFF	ISO/IEC 14443-3:2018, 7.1.7			
10	Deactivation frame waiting time (TR0+TR1)	ISO/IEC 14443-3:2018, 7.1.6, ISO/IEC 14443-4:2018, 8.1			

Table G.112 — Reported results for Type A specific test methods

Test method from ISO/IEC 10373-6		Scenario numbers	Test result
Clause	Parameter	ISO/IEC 10373-6	PASS or FAIL or N/A
G.3.2	Polling	Scenario G.1	
G.3.3	Testing of the PICC Type A state transitions	Scenario G.2	
		Scenario G.3	
		Scenario G.4	
		Scenario G.5	
		Scenario G.6	
		Scenario G.7	
		Scenario G.8	
		Scenario G.9	
		Scenario G.10	
		Scenario G.11	
		Scenario G.12	
G.3.4	Handling of Type A anticollision	Scenario G.13	
G.3.6	Handling of PPS request	Scenario G.17	
		Scenario G.18	
		Scenario G.19	
G.3.7	Handling of FSD	Scenario G.20	
G.3.8	Handling of Frame Delay Time PICC to PCD and SFGT	Scenario G.71	
G.3.9	PICC bit rates capability	Scenario G.72	

Table G.113 — Reported results for Type B specific test methods

Test method from ISO/IEC 10373-6		Scenario numbers	Test result
Clause	Parameter	ISO/IEC 10373-6	PASS or FAIL or N/A
G.4.2	Polling	Scenario G.21	
G.4.3	PICC framing and bit rates capability	Scenario G.22	
G.4.4	Testing of the PICC Type B state transitions	Scenario G.23	
		Scenario G.24	
		Scenario G.25	
		Scenario G.26	
		Scenario G.27	
G.4.5	Handling of Type B anticollision	Scenario G.28	
G.4.6	Handling of ATTRIB	Scenario G.29	
		Scenario G.30	
G.4.7	Handling of Maximum Frame Size	Scenario G.31	
G.4.8	Handling of TR2 and SFGT	Scenario G.73	

Table G.114 — Reported results for test methods for logical operation of the PICC Type A or Type B

Test method from ISO/IEC 10373-6		Scenario numbers	Test result
Clause	Parameter	ISO/IEC 10373-6	PASS or FAIL or N/A
G.5.2	PICC reaction to ISO/IEC 14443-4 scenarios	Scenario G.32	
		Scenario G.33	
		Scenario G.34	
		Scenario G.35	
		Scenario G.36	
		Scenario G.37	
		Scenario G.38	
		Scenario G.39	
		Scenario G.40	
		Scenario G.41	
		Scenario G.42	
		Scenario G.43	
		Scenario G.74	
		Scenario G.44	
		Scenario G.45	
		Scenario G.46	
		Scenario G.47	
		Scenario G.48	
		Scenario G.49	
		Scenario G.50	
		Scenario G.51	
		Scenario G.52	
		Scenario G.53	
		Scenario G.54	
G.5.3	Handling of PICC error detection	Scenario G.55	
		Scenario G.56	
		Scenario G.57	
G.5.4	PICC reaction on CID	Scenario G.58	
		Scenario G.59	
		Scenario G.60	
		Scenario G.61	
		Scenario G.62	
		Scenario G.66	
G.5.5	PICC reaction on NAD	Scenario G.63	
		Scenario G.64	
		Scenario G.65	
G.5.6	PICC reaction on S(PARAMETERS) blocks	Scenario G.67	
		Scenario G.68	
		Scenario G.69	
		Scenario G.70	

Table G.114 *(continued)*

Test method from ISO/IEC 10373-6		Scenario numbers	Test result
Clause	**Parameter**	**ISO/IEC 10373-6**	**PASS or FAIL or N/A**
G.5.7	PICC supporting Type A and Type B		

Annex H
(normative)

Additional PCD test methods

H.1 PCD-test-apparatus and accessories

H.1.1 General

This clause defines the PCD-test-apparatus and test circuits for verifying the operation of the PCD according to ISO/IEC 14443-3 and ISO/IEC 14443-4.

H.1.2 Test method

The ISO/IEC 9646 abstract model is chosen and the local test method is used for the testing of the ISO/IEC 14443 protocol between the tested PCD and the LT.

H.1.3 PCD-test-apparatus structure

The PCD-test-apparatus consists of two parts (see Figure H.1):

1) Upper Tester (may be a personal computer with a host interface suitable for a tested PCD);

2) Lower Tester (LT).

Tested PCD is treated as Implementation Under Test (IUT).

When a PCD is embedded in a product, it includes the UT. For this case some tests may not be applicable. Also, in case the standard does not have a specific requirement the test method will end up in a report of capabilities only.

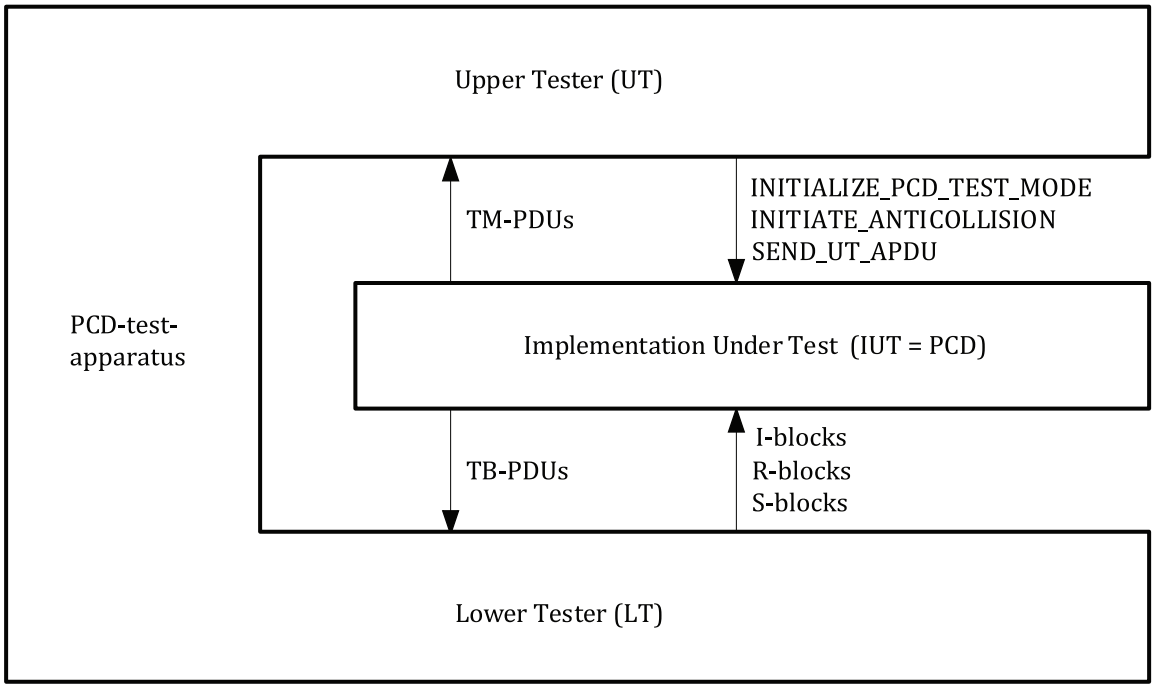


Figure H.1 — Conceptual tester architecture

The LT part of the PCD-test-apparatus includes:

1) a PICC emulation hardware and software device capable of emulating of Type A and Type B protocols;

2) a digital sampling oscilloscope (see 5.2).

H.1.4 PCD-test-apparatus interface

The UT and the IUT communicate with the TM-PDU (Test Management PDU). The definition of TM-PDUs is implementation dependent and provided by the IUT manufacturer and shall initiate the actions required in Table H.1.

Table H.1 — Logical interface commands

No.	TM-PDU name	Required IUT action
1	INITIALIZE_PCD_TEST_MODE	Return to Power On state (The IUT is expected to enter to anticollision loop). The IUT returns the result code of its action to the UT.
2	INITIATE_ANTICOLLISION	Initiate anticollision sequence (if the IUT starts the anticollision sequence automatically upon initialize, the sequence can be empty). The IUT returns the result code of its action to the UT.
3	SEND_UT_APDU	Transmit the UT_APDU through the RF interface to LT and return the IUT result code of its action to the UT. The response from the IUT shall include the answer of LT to the UT_APDU sent.

The PCD-test-apparatus shall be able to initialize the IUT utility information provided by the IUT manufacturer over the UT interface and to configure itself to perform the necessary procedures, protocols and analysis over its LT interface.

H.1.5 Emulating the I/O protocol

The PCD-test-apparatus at its LT interface shall be able to emulate the protocol Type A and Type B and PICC applications, which are required to run the scenario. The LT shall be able to break the transmitted packets into chained blocks with the required length.

It shall be possible to configure the LT to simulate different options:

1) NAD and CID configuration;

2) frame size, bit rates and any other parameter as required for the implementation of the test methods.

H.1.6 Generating the I/O character timing in transmission mode

The PCD-test-apparatus at its LT interface shall be able to generate the I/O bit stream according to ISO/IEC 14443-3. Timing parameters: start bit duration, Extra Guard Time EGT (Type B only), bit duration, Frame Delay Time, start of frame width and end of frame width shall be configurable.

For the purpose of tests of Type A, the LT shall be capable of simulating a bit collision at a selected bit position(s).

H.1.7 Measuring and monitoring the RF I/O protocol

H.1.7.1 Timing measurements

The PCD-test-apparatus shall continuously monitor the following frame format and timing values:

Type A:

1) Frame Delay Time PICC to PCD (see ISO/IEC 14443-3:2018, 6.2.1.2);

2) frame formats (see ISO/IEC 14443-3:2018, 6.2.3);

Type B:

1) character, frame format and timing (see ISO/IEC 14443-3:2018, 7.1).

A test shall be FAIL and the tested PCD shall be declared non-compliant in case one of the listed timing constraints is violated.

H.1.7.2 Timing measurement report

Fill out Table H.56 for Type A and Table H.57 for Type B with the measured timing values.

H.1.8 Protocol analysis

The PCD-test-apparatus shall be able to analyze the I/O-bit stream at its LT interface in accordance with protocol Type A and Type B as specified in ISO/IEC 14443-3 and ISO/IEC 14443-4 and extract the logical data flow for further protocol analysis.

H.1.9 Protocol activation procedure

H.1.9.1 Activation procedure for anticollision test methods

Activate the LT by the following sequence:

a) Configure the LT to emulate the Type A or Type B protocol.

b) The UT sends INITIALIZE_PCD_TEST_MODE TM-PDU to the PCD.

c) The UT sends INITIATE_ANTICOLLISION TM-PDU to the PCD.

H.1.9.2 Activation procedure for Type A protocol test methods

Activate the LT by the following sequence:

a) Configure the LT to emulate the Type A protocol.

b) The UT sends INITIALIZE_PCD_TEST_MODE TM-PDU to the PCD.

c) The UT sends INITIATE_ANTICOLLISION TM-PDU to the PCD. The PCD shall apply the anticollision sequence as defined in ISO/IEC 14443-3:2018, Clause 6 (request, anticollision loop and select). The PCD shall apply the protocol activation sequence as defined in ISO/IEC 14443-4:2018, Clause 5.

d) The PCD reports to the UT the result of the activation procedure.

H.1.9.3 Activation procedure for Type B protocol test methods

Activate the LT by the following sequence:

a) Configure the LT to emulate the Type B protocol.

b) The UT sends INITIALIZE_PCD_TEST_MODE TM-PDU to the PCD.

c) The UT sends INITIATE_ANTICOLLISION TM-PDU to the PCD. The PCD shall apply the anticollision sequence as defined in ISO/IEC 14443-3:2018, Clause 7.

d) The PCD reports to the UT the result of the activation procedure.

H.1.10 Scenario

H.1.10.1 Description

Testing of the PCD as defined in this document requires a scenario to be executed. This scenario is a 'typical protocol and application specific communication', dependent on the protocol and application specific functionality foreseen for the normal use of and implemented in the PCD.

The typical scenario is the set of command TM-PDUs defined in H.1.4.

The scenario shall be defined by the entity carrying out these tests and shall be documented with the test results. The scenario shall encompass a representative subset or, if practical, the full functionality of the PCD expected to be utilized during normal use.

NOTE The testing entity may require information about the implemented protocol and functionality.

The UT_APDU to be sent may be one from the following:

1) UT_TEST_COMMAND1, decided by the PCD-test-apparatus, specifies the ISO instruction used as the default instruction for scenarios not needing PCD chaining. (In case PCD decides anyway to chain, the scenario should be adapted accordingly by the test laboratory);

2) UT_TEST_COMMAND2, decided by the PCD-test-apparatus, specifies the ISO instruction used as the default instruction for scenarios dealing with PCD chaining.

H.1.10.2 Scenario example

The typical scenario may be as follows:

```
INITIALIZE_PCD_TEST_MODE
INITIATE_ANTICOLLISION
SEND_UT_APDU (UT_TEST_COMMAND1)
SEND_UT_APDU (UT_TEST_COMMAND2)
….
```

H.1.11 UT, LT and PCD behavior

The following items summarize the behavior of the UT, the LT and the PCD:

a) The UT runs the activation procedure as defined in H.1.9.

b) If the activation procedure went wrong, the PCD goes to exception processing. This exception processing may include reporting the error to the UT.

c) In case of anticollision test methods, the PCD-test-apparatus ends the test at this point. For protocol test methods the UT continues to the next step.

d) The UT sends the first command UT_APDU to the PCD.

e) The PCD is expected to transfer this command UT_APDU to the LT using TB-PDUs. The PCD splits the current UT_APDU into the appropriate TB-PDUs, sends the first block to the LT and awaits the response block. The PCD manages communication blocks according to ISO/IEC 14443-4, which may also include S(PARAMETERS) exchange.

f) The command UT_APDU is received by the LT. The LT sends the response UT_APDU to the PCD. The LT manages communication blocks (TB-PDUs) according to ISO/IEC 14443-4 (the LT may use chaining mechanism at any time even if not mandated by either PCD or PICC maximum frame size). The PCD is expected to transfer response UT_APDU, received from the LT, back to the UT.

g) If the command failed at protocol level (i.e. error detected by the PCD), the PCD goes to exception processing. Exception processing may include error reporting to the UT.

h) If the command succeeded, the PCD reports the UT about successful result. In this case, if the scenario defines additional UT_APDU to be sent to the LT, the UT sends the next UT_APDU to the PCD. This loop continues until the last test UT_APDU is sent.

H.1.12 Relationship of test methods versus base standard requirement

All tests in Table H.2, Table H.3 and Table H.4 shall be executed and their results reported in the relevant tables in H.6.

Table H.2 — Type A specific test methods

Test method from ISO/IEC 10373-6		Corresponding requirement	
Clause	**Name**	**Base standard**	**Clause**
H.2.1	Frame Delay Time PICC to PCD	ISO/IEC 14443-3:2018	6.2.1.2
H.2.2	Request Guard Time	ISO/IEC 14443-3:2018	6.2.2
H.2.3	Handling of bit collision during ATQA	ISO/IEC 14443-3:2018	6.5.2
H.2.4	Handling of anticollision loop	ISO/IEC 14443-3:2018	6.5.3
H.2.5	Handling of RATS and ATS	ISO/IEC 14443-4:2018	5.6.1.1
H.2.6	Handling of PPS response	ISO/IEC 14443-4:2018	5.6.2.1
H.2.7	Frame size selection mechanism	ISO/IEC 14443-4:2018	5.2.3
H.2.8	Handling of Start-up Frame Guard Time	ISO/IEC 14443-4:2018	5.2.5
H.2.9	Handling of the CID during activation by the PCD	ISO/IEC 14443-4:2018	5.6.3
H.2.10	Handling of parity bit	ISO/IEC 14443-3:2018	6.2.3.2

Table H.3 — Type B specific test methods

Test method from ISO/IEC 10373-6		Corresponding requirement	
Clause	**Name**	**Base standard**	**Clause**
H.3.2	Frame size selection mechanism	ISO/IEC 14443-3:2018	7.9
H.3.3	Handling of the CID during activation by the PCD	ISO/IEC 14443-3:2018	7.10
H.3.4	Frame Delay Time PICC to PCD (TR2)	ISO/IEC 14443-3:2018	7.9.4.4
H.3.5	Handling of Start-up Frame Guard Time	ISO/IEC 14443-3:2018	7.9.4.7
H.3.6	Type B PCD framing tests	ISO/IEC 14443-3:2018	7.1

Table H.4 — Test methods for logical operation

Test method from ISO/IEC 10373-6		Corresponding requirement	
Clause	**Name**	**Base standard**	**Clause**
H.4.2	Handling of the polling loop	ISO/IEC 14443-3:2018	Clause 5
H.4.3	Reaction of the PCD to request for waiting time extension	ISO/IEC 14443-4:2018	7.3
H.4.4	Error detection and recovery	ISO/IEC 14443-4:2018	7.5.7
H.4.5	Handling of NAD during chaining	ISO/IEC 14443-4:2018	7.1.2.3

H.2 Type A specific test methods

H.2.1 Frame Delay Time PICC to PCD

H.2.1.1 Scope

The purpose of this test is to determine the timing between a PICC frame and the next PCD frame.

H.2.1.2 Apparatus

See H.2.

H.2.1.3 Procedure

Place the LT into the PCD operating volume.

During the following procedure the RF Input/Receive data shall be continuously monitored and verified correct to ISO/IEC 14443-2. All signal transitions (level and timing) as well as the logical content of the communication shall be recorded.

Use the following sequence:

a) The UT performs the activation procedure according to H.1.9.1.

b) The LT waits until the PCD sends a valid REQA/WUPA command frame.

c) The LT answers with ATQA indicating bit frame anticollision and UID size: single (bits b8 and b7 equal (00)b).

d) The PCD shall send ANTICOLLISION command '93 20' (cascade level 1).

e) The LT answers with UID CL1 (uid0 uid1 uid2 uid3 BCC).

f) The PCD shall send SELECT command '93 70' uid0 uid1 uid2 uid3 BCC CRC_A.

g) The LT answers with SAK indicating that UID is complete (b3 = (0)b) and that PICC is compliant with ISO/IEC 14443-4 (b6 = (1)b).

h) The PCD shall send a valid RATS command frame.

i) The LT answers with a valid ATS (T0 = '70', TA(1)='77', TB(1) = 'A0', TC(1) = '02', no historical bytes).

j) The PCD may send a PPS request. In that case, the LT answers with a valid PPS response.

k) The PCD may send a S(PARAMETERS) request. In that case, the LT answers with S(PARAMETERS) response.

l) The PCD shall return the result code, in accordance with Table H.1, of its action to the UT.

m) The UT sends the SEND_UT_APDU(UT_TEST_COMMAND1) to the PCD.

n) The LT waits until the PCD sends an I-block to the LT with the INF field containing the UT_TEST_COMMAND1.

o) The LT sends I-block (containing some response UT_APDU with answer to the UT_TEST_COMMAND1) to the PCD.

p) The PCD is expected to transfer the response UT_APDU back to the UT. Check at the UT that this response UT_APDU block is correctly accepted.

q) The UT sends the SEND_UT_APDU(UT_TEST_COMMAND1) to the PCD.

r) The LT waits until the PCD sends an I-block to the LT with the INF field containing the UT_TEST_COMMAND1.

s) The LT sends S(WTX) request using any valid WTXM.

t) The PCD shall send S(WTX) response using the same WTXM as in the S(WTX) request. If it does not meet the expected response, end the test at this point.

u) The LT sends I-block (containing some response UT_APDU with answer to the UT_TEST_COMMAND1) to the PCD.

v) Measure the time between the last modulation transmitted by the LT and the first pause transmitted by the PCD (see ISO/IEC 14443-3:2018, 6.2.1.2) for each command.

Table H.5 gives part of the procedure as a scenario.

Table H.5 — Scenario H.27: Frame Delay Time PICC to PCD

PCD		LT
REQA/WUPA	→	
	←	ATQA (single size UID)
ANTICOLLISION command Level 1 ('93 20')	→	
	←	UID CL1 (uid0 uid1 uid2 uid3 BCC)
SELECT command ('93 70' uid0 uid1 uid2 uid3 BCC CRC_A)	→	
	←	SAK (UID complete, PICC compliant with ISO/IEC 14443-4)
RATS command frame (e.g.'E0 01' CRC_A)	→	
	←	ATS (T0= '77', TA(1)='77', TB(1)='A0', CID supported, no historical bytes)
PPS request	→	
	←	PPS response
S(PARAMETERS)	→	
	←	S(PARAMETERS)
$I(0)_0$(INF = UT_TEST_COMMAND1)	→	
	←	$I(0)_0$(INF = answer to UT_TEST_COMMAND1)
$I(0)_1$(INF = UT_TEST_COMMAND1)	→	
	←	S(WTX) request
S(WTX) response	→	
	←	$I(0)_1$(INF = answer to UT_TEST_COMMAND1)
NOTE If any of the steps requires command sent by the UT, the subsequent steps may be skipped.		

H.2.1.4 Test report

Report the signal recording. Fill item 1 of Table H.56 with the minimum measured value of Frame Delay Time PICC to PCD and the appropriate row of Table H.58 in accordance with Table H.6.

Table H.6 — Result criteria for Frame Delay Time PICC to PCD

Explanation	Test result
Only when the PCD's Frame Delay Time PICC to PCD always exceed the minimum value defined in ISO/IEC 14443-3:2018, 6.2.1.2.	PASS
Any other case	FAIL

H.2.2 Request Guard Time

H.2.2.1 Scope

The purpose of this test is to determine the Request Guard Time of two consecutive REQA/WUPA commands. This test is relevant for PCDs, which send consecutive REQA/WUPA.

H.2.2.2 Apparatus

See H.2.

H.2.2.3 Procedure

Place the LT into the PCD operating volume.

During the following procedure, the RF Input/Receive data shall be continuously monitored and verified correct according to ISO/IEC 14443-2. All signal transitions (level and timing) as well as the logical content of the communication shall be recorded.

Use the following sequence:

a) The UT performs the activation procedure according to H.1.9.1.

b) The LT waits until the PCD sends a valid REQA/WUPA command frame. The LT remains Mute.

c) The LT waits until the PCD sends a valid REQA/WUPA command frame. The LT remains Mute.

d) Measure the time between the start bits of two consecutive REQA/WUPA (see ISO/IEC 14443-3:2018, 6.2.2).

H.2.2.4 Test report

Report the signal recording. Fill item 2 in Table H.56 with measured value of Request Guard Time and the appropriate row of Table H.58 in accordance with Table H.7.

Table H.7 — Result criteria for Request Guard Time

Explanation	Test result
Only when the PCD's Request Guard Time always exceed the minimum value defined in ISO/IEC 14443-3:2018, 6.2.2.	PASS
Any other case	FAIL

H.2.3 Handling of bit collision during ATQA

H.2.3.1 Scope

The purpose of this test is to determine the handling of bit collision during ATQA by the PCD.

H.2.3.2 Apparatus

See H.1.

H.2.3.3 Procedure

Place the LT into the PCD operating volume and record the presence and the content of the PCD commands.

Use the following sequence:

a) The UT performs the activation procedure according to H.1.9.1.

b) The LT waits until the PCD sends a valid REQA/WUPA command frame.

c) Maintain the LT to answer with ATQA using simulation of the bit collision at bit N (N from 1 up to 16). Collision at a bit causes a collision also in associated parity bit.

H.2.3.4 Test report

Fill the appropriate row in Table H.58 according to Table H.8.

Table H.8 — Result criteria for handling of bit collision during ATQA

Explanation	Test result
Only when the PCD starts the bit oriented anticollision loop	PASS
Any other case	FAIL

H.2.4 Handling of anticollision loop

H.2.4.1 Scope

The purpose of this test is to determine the handling of bit anticollision loop according to ISO/IEC 14443-3:2018, 6.5.3.

H.2.4.2 Apparatus

See H.1.

H.2.4.3 Procedure

H.2.4.3.1 General

Place the LT into the PCD operating volume and record the presence and the content of the PCD commands.

H.2.4.3.2 Procedure 1 (single size UID)

Use the following sequence:

a) The UT performs the activation procedure according to H.1.9.1.

b) The LT waits until the PCD sends a valid REQA/WUPA command frame.

c) The LT answers with ATQA indicating bit frame anticollision and UID size: single (bits b8 and b7 equal (00)b).

d) The PCD shall send ANTICOLLISION command '93 20' (cascade level 1).

e) The LT answers with UID CL1 (uid0 uid1 uid2 uid3 BCC).

f) The PCD shall send SELECT command '93 70' uid0 uid1 uid2 uid3 BCC CRC_A.

g) The LT answers with SAK (cascade bit is cleared, b3 = (0)b), indicating that UID is complete.

Table H.9 gives part of the procedure as a scenario.

Table H.9 — Scenario H.1: Handling of anticollision loop for PICC with single size UID (Procedure 1)

Test	PCD		LT	Stage
REQA/WUPA	REQA/WUPA	→		
		←	ATQA (single size UID)	1

Table H.9 *(continued)*

Test	PCD		LT	Stage
ANTICOLLISION Level 1	ANTICOLLISION command Level 1 ('93 20')	→		
		←	UID CL1 (uid0 uid1 uid2 uid3 BCC)	2
SELECT	SELECT command ('93 70' uid0 uid1 uid2 uid3 BCC CRC_A)	→		
		←	SAK(complete)	3

H.2.4.3.3 Procedure 2 (double size UID)

Use the following sequence:

a) The UT performs the activation procedure according to H.1.9.1.

b) The LT waits until the PCD sends a valid REQA/WUPA command frame.

c) The LT answers with ATQA indicating bit frame anticollision and UID size: double (bits b8 and b7 equal (01)b).

d) The PCD shall send ANTICOLLISION command '93 20' (cascade level 1).

e) The LT answers with UID CL1 ('88' uid0 uid1 uid2 BCC).

f) The PCD shall send SELECT command '93 70 88' uid0 uid1 uid2 BCC CRC_A.

g) The LT answers with SAK (cascade bit is set, b3 = (1)b).

h) The PCD shall increase the cascade level and shall send ANTICOLLISION command '95 20' (cascade level 2).

i) The LT answers with UID CL2 (uid3 uid4 uid5 uid6 BCC).

j) The PCD shall send SELECT command '95 70' uid3 uid4 uid5 uid6 BCC CRC_A.

k) The LT answers with SAK (cascade bit is cleared, b3 = (0)b), indicating that UID is complete.

Table H.10 gives part of the procedure as a scenario.

Table H.10 — Scenario H.2: Handling of anticollision loop for PICC with double size UID (Procedure 2)

Test	PCD		LT	Stage
REQA/WUPA	REQA/WUPA	→		
		←	ATQA (double size UID)	1
ANTICOLLISION Level 1	ANTICOLLISION command Level 1 ('93 20')	→		
		←	UID CL1 ('88' uid0 uid1 uid2 BCC)	2
SELECT	SELECT command ('93 70 88' uid0 uid1 uid2 BCC CRC_A)	→		
		←	SAK(cascade)	3

Table H.10 *(continued)*

Test	PCD		LT	Stage
ANTICOLLISION Level 2	ANTICOLLISION command Level 2 ('95 20')	→		
		←	UID CL2 (uid3 uid4 uid5 uid6 BCC)	4
SELECT	SELECT command ('95 70' uid3 uid4 uid5 uid6 BCC CRC_A)	→		
		←	SAK(complete)	5

H.2.4.3.4 Procedure 3 (triple size UID)

Use the following sequence:

a) The UT performs the activation procedure according to H.1.9.1.

b) The LT waits until the PCD sends a valid REQA/WUPA command frame.

c) The LT answers with ATQA indicating bit frame anticollision and UID size: triple (bits b8 and b7 equal (10)b).

d) The PCD shall send ANTICOLLISION command '93 20' (cascade level 1).

e) The LT answers with UID CL1 ('88' uid0 uid1 uid2 BCC).

f) The PCD shall send SELECT command '93 70 88' uid0 uid1 uid2 BCC CRC_A.

g) The LT answers with SAK (cascade bit is set, b3 = (1)b).

h) The PCD shall increase the cascade level and shall send ANTICOLLISION command '95 20' (cascade level 2).

i) The LT answers with UID CL2 ('88' uid3 uid4 uid5 BCC).

j) The PCD shall send SELECT command '95 70 88' uid3 uid4 uid5 BCC CRC_A.

k) The LT answers with SAK (cascade bit is set, b3 = (1)b).

l) The PCD shall increase the cascade level and shall send ANTICOLLISION command '97 20' (cascade level 3).

m) The LT answers with UID CL3 (uid6 uid7 uid8 uid9 BCC).

n) The PCD shall send SELECT command '97 70' uid6 uid7 uid8 uid9 BCC CRC_A.

o) The LT answers with SAK (cascade bit is cleared, b3 = (0)b), indicating that UID is complete.

Table H.11 gives part of the procedure as a scenario.

Table H.11 — Scenario H.3: Handling of anticollision loop for PICC with triple size UID (Procedure 3)

Test	PCD		LT	Stage
REQA/WUPA	REQA/WUPA	→		
		←	ATQA (triple size UID)	1

Table H.11 *(continued)*

Test	PCD		LT	Stage
ANTICOLLISION Level 1	ANTICOLLISION command Level 1 ('93 20')	→		
		←	UID CL1 ('88' uid0 uid1 uid2 BCC)	2
SELECT	SELECT command ('93 70 88' uid0 uid1 uid2 BCC CRC_A)	→		
		←	SAK(cascade)	3
ANTICOLLISION Level 2	ANTICOLLISION command Level 2 ('95 20')	→		
		←	UID CL2 ('88' uid3 uid4 uid5 BCC)	4
SELECT	SELECT command ('95 70 88' uid3 uid4 uid5 BCC CRC_A)	→		
		←	SAK(cascade)	5
ANTICOLLISION Level 3	ANTICOLLISION command Level 3 ('97 20')	→		
		←	UID CL3 (uid6 uid7 uid8 uid9 BCC)	6
SELECT	SELECT command ('97 70' uid6 uid7 uid8 uid9 BCC CRC_A)	→		
		←	SAK(complete)	7

H.2.4.3.5 Procedure 4 (Full Bitwise Anticollision, single size UID)

Use the following sequence:

a) The UT performs the activation procedure according to H.1.9.1.

b) The LT waits until the PCD sends a valid REQA or WUPA command frame.

c) The LT answers with ATQA indicating bit frame anticollision and UID size: single (bits b8 and b7 equal (00)b).

d) The PCD shall send ANTICOLLISION command: '93 20'.

e) The LT answers by a stream of 40 bits by emulating a collision on every bit, including parity bits.

f) Repeat the steps g) to h) for values k from 1 to 31.

g) The PCD shall send ANTICOLLISION command: '93' NVB $UIDTX_1[[1..k]]$, where $UIDTX_1[[1..k–1]]$ is either empty (i.e. k = 1) or the value already known by the PCD and $UIDTX_1[[k]]$ is an arbitrary bit selected by the PCD.

h) The LT answers by a stream of 40 minus k bits by emulating a collision on every bit, including parity bits.

i) The PCD may optionally send ANTICOLLISION command: '93 60' $UIDTX_1[[1..32]]$. In this case the LT answers with BCC.

NOTE This optional ANTICOLLISON command does not change the Stage number.

j) The PCD shall send SELECT command '93 70' $UIDTX_1$[[1..32]] BCC CRC_A. Note, that the PCD has to calculate the BCC if it has not run the optional step i).

k) The LT answers with SAK (cascade bit is cleared, b3 = (0)b), indicating that UID is complete.

Table H.12 gives part of the procedure as a scenario.

Table H.12 — Scenario H.4: Handling of full bitwise anticollision loop for PICC (Procedure 4)

Test	PCD		LT	Stage
REQA/WUPA	REQA/WUPA	→		
		←	ATQA (single size UID)	1
ANTICOLLISION	ANTICOLLISION command ('93 20')	→		
		←	40 bits full collision frame	2
ANTICOLLISION (k bits $UID_{PARTIAL}$) $1 \le k \le 31$	ANTICOLLISION command ('93' NVB $UIDTX_1$[[1..k]])	→		
		←	40 minus k bits collision frame	k+2
OPTIONAL ANTICOLLISION (32 bits $UID_{PARTIAL}$)	ANTICOLLISION command ('93 60' $UIDTX_1$[[1..32]])	→		
		←	(BCC)	
SELECT	SELECT command ('93 70' $UIDTX_1$[[1..32]] BCC CRC_A)	→		
		←	SAK(complete)	34

H.2.4.4 Test report

Fill the appropriate row in Table H.58 according to Table H.13.

Table H.13 — Result criteria for handling of anticollision loop

Explanation	Test result
Only when the PCD's behavior matches each procedure expected scenario	PASS
Any other case	FAIL

H.2.5 Handling of RATS and ATS

H.2.5.1 Scope

The purpose of this test is to determine the handling of RATS and ATS by the PCD according to ISO/IEC 14443-4:2018, 5.6.1.1.

H.2.5.2 Apparatus

See H.1.

H.2.5.3 Procedure

H.2.5.3.1 General

Place the LT into the PCD operating volume and record the presence and the content of the PCD commands.

H.2.5.3.2 Procedure 1

Use the following sequence:

a) The UT performs the activation procedure according to H.1.9.1.

b) The LT answers relevant anticollision messages and waits until the PCD sends a valid RATS command frame.

c) The LT does not respond to RATS (Mute).

d) The PCD may send a valid RATS command frame.

e) If the PCD has sent a second RATS, the LT does not respond to the RATS (Mute).

f) The PCD shall start the deactivation sequence defined in ISO/IEC 14443-4:2018, Clause 8.

g) Repeat the steps a) to f) with an erroneous ATS frame (use a wrong CRC_A) instead of Mute in step c).

h) Repeat the steps a) to f) with an erroneous ATS frame (use a wrong parity bit) instead of Mute in step c).

Table H.14 gives part of the procedure as a scenario.

Table H.14 — Scenario H.5: Handling of RATS and ATS, Procedure 1

Test	PCD		LT
Mute	RATS command frame (e.g. 'E0 01' CRC_A)	→	
		←	Mute
	RATS command frame (e.g. 'E0 01' CRC_A)	→	
		←	Mute
Start deactivation	DESELECT	→	
Erroneous ATS frame (wrong CRC_A)	RATS command frame (e.g. 'E0 01' CRC_A)	→	
		←	Erroneous ATS frame (wrong CRC_A)
	RATS command frame (e.g. 'E0 01' CRC_A)	→	
		←	Mute
Start deactivation	DESELECT	→	
Erroneous ATS frame (wrong parity bit)	RATS command frame (e.g. 'E0 01' CRC_A)	→	
		←	Erroneous ATS frame (wrong parity bit)[a]
	RATS command frame (e.g. 'E0 01' CRC_A)	→	
		←	Mute
Start deactivation	DESELECT	→	

[a] The parity error is simulated on the first transmitted byte of the frame by reversing the parity bit.

H.2.5.3.3 Procedure 2

Use the following sequence:

a) The UT performs the activation procedure according to H.1.9.1.

b) The LT answers relevant anticollision messages and waits until the PCD sends a valid RATS command frame.

c) The LT answers with a valid ATS without TA byte.

d) The PCD shall return the result code, in accordance with Table H.1, of its action to the UT.

e) The UT sends the SEND_UT_APDU(UT_TEST_COMMAND1) to the PCD.

f) The PCD is expected to send any I-block (including empty) to the LT, possibly after PICC presence check sequences.

Table H.15 gives part of the procedure as a scenario.

Table H.15 — Scenario H.6: Handling of RATS and ATS, Procedure 2

Test	PCD		LT
Correct ATS	RATS command frame (e.g.'E0 01' CRC_A)	→	
		←	ATS
Continue operation	Any I-block (including empty)	→	

H.2.5.3.4 Void

Procedure 3, which was defined in in former editions of this document, is no longer present.

H.2.5.3.5 Procedure 4

Use the following sequence:

a) The UT performs the activation procedure according to H.1.9.1.

b) The LT waits until the PCD sends a valid REQA/WUPA command frame.

c) The LT answers with ATQA indicating bit frame anticollision and UID size: single (bits b8 and b7 equal (00)b) and proprietary bits b12 to b9 set to (0000)b.

d) The PCD shall send ANTICOLLISION command '93 20' (cascade level 1).

e) The LT answers with UID CL1 (uid0 uid1 uid2 uid3 BCC).

f) The PCD shall send SELECT command '93 70' uid0 uid1 uid2 uid3 BCC CRC_A.

g) The LT answers with SAK (b3 = (0)b, b6=(1)b, all other bits are set to (0)b).

h) The PCD shall send a valid RATS command frame.

i) Repeat the procedure using SAK (b3 = (0)b, b6=(1)b, all other bits are set to (1)b) at step g) and using ATQA with proprietary bits b12 to b9 set to (1111)b at step a).

Table H.16 gives part of the procedure as a scenario.

Table H.16 — Scenario H.30: Handling of RATS and ATS, Procedure 4

PCD		LT
REQA/WUPA	→	
	←	ATQA (single size UID, proprietary bits b12 to b9 set to (0)b) or ATQA (single size UID, proprietary bits b12 to b9 set to (1)b)[a]
ANTICOLLISION command Level 1 ('93 20')	→	
	←	UID CL1 (uid0 uid1 uid2 uid3 BCC)
SELECT command ('93 70' uid0 uid1 uid2 uid3 BCC CRC_A)	→	
	←	SAK(b3=(0)b, b6=(1)b, all other bits are set to (0)b) or SAK(b3=(0)b, b6=(1)b, all other bits are set to (1)b)[a]
RATS command frame (e.g.'E0 01' CRC_A)[b]	→	

[a] Determined in step i).

[b] The PCD may send "proprietary" commands before sending RATS.

Scenario H.7: Void

Scenario H.7, Handling of RATS and ATS, Procedure 3, which was defined in former editions of this document, is no longer present.

H.2.5.4 Test report

Fill the appropriate row in Table H.58 according to Table H.17.

Table H.17 — Result criteria for handling of RATS and ATS

Explanation	Test result
Only when the PCD's behavior matches each procedure expected scenario	PASS
Any other case	FAIL

H.2.6 Handling of PPS response

Handling of PPS response is tested in I.2.1.

Scenario H.8: Void

Scenario H.8, Handling of PPS request and response, Procedure 1, which was defined in former editions of this document, is no longer present.

Scenario H.9: Void

Scenario H.9, Handling of PPS request and response, Procedure 2, which was defined in former editions of this document, is no longer present.

H.2.7 Frame size selection mechanism

H.2.7.1 Scope

The purpose of this test is to verify the correct handling of transmitted frame size. The transmitted frames shall not be longer than indicated by FSCI.

This test shall be executed for at least FSCI set to '0', '1', '8' and 'C'.

H.2.7.2 Apparatus

See H.1.

H.2.7.3 Procedure

Place the LT into the PCD operating volume and record the presence and the content of the PCD commands.

Use the following sequence:

a) The UT performs the activation procedure according to H.1.9.1.

b) The LT answers relevant anticollision messages and waits until the PCD sends a valid RATS command frame.

c) The LT answers with a valid ATS. For the purpose of this test, the LT returns format byte T0 equal '70' (see ISO/IEC 14443-4:2018, 5.2.3). In case there is a PPS request the LT will answer it before continuing with the next step.

d) The UT sends the SEND_UT_APDU(UT_TEST_COMMAND2) to the PCD where the data length shall be more than the maximum size of a frame accepted by the LT.

e) The PCD shall send the following $I(1)_0$ block with maximum length of in accordance with FSCI.

Table H.18 gives part of the procedure as a scenario.

Table H.18 — Scenario H.10: Frame size selection mechanism

PCD		LT
RATS command frame (e.g. 'E0 01' CRC_A)	→	
	←	ATS, T0 = '70'
Optional PPS request is possible and would be processed by the LT before the I-block	→	
	←	PPS response in case of PPS request
$I(1)_0$ (maximum length in accordance with FSCI)	→	

H.2.7.4 Test report

Fill the appropriate row in Table H.58 according to Table H.19.

Table H.19 — Result criteria for frame size selection mechanism

Explanation	Test result
Only when the PCD's behavior matches the expected scenario for all tested FSCI values	PASS
Any other case	FAIL

H.2.8 Handling of Start-up Frame Guard Time

H.2.8.1 Scope

The purpose of this test is to determine the PCD transmission timing according to ISO/IEC 14443-4:2018, 5.2.5.

This test shall be executed for at least SFGI set to 0, 1 and 14.

H.2.8.2 Apparatus

See H.1.

H.2.8.3 Procedure

Place the LT into the PCD operating volume and record the presence and the content of the PCD commands.

During the following procedure the RF Input/Receive data shall be continuously monitored and verified correct to ISO/IEC 14443-2. All signal transitions (level and timing) as well as the logical content of the communication shall be recorded.

Use the following sequence:

a) The UT performs the activation procedure according to H.1.9.1.

b) The LT answers relevant anticollision messages and waits until the PCD sends a valid RATS command frame.

c) The LT answers with a valid ATS.

d) The UT sends the SEND_UT_APDU(UT_TEST_COMMAND1) to the PCD.

e) The PCD shall send next block or a PPS request after a minimum delay of SFGT.

Table H.20 gives part of the procedure as a scenario.

Table H.20 — Scenario H.11: Start-up Frame Guard Time mechanism

PCD		LT
RATS command frame (e.g. 'E0 01' CRC_A)	→	
	←	ATS
PPS Request or next block (Sent after SFGT = $(256 \times 16/f_c) \times 2^{SFGI}$)	→	

H.2.8.4 Test report

Fill item 3 of Table H.56 with measured values and fill the appropriate row of Table H.58 in accordance with Table H.21.

Table H.21 — Result criteria for handling of Start-up Frame Guard Time

Explanation	Test result
Only when the PCD's behavior matches the expected scenario for all tested SFGI values	PASS
Any other case	FAIL

H.2.9 Handling of the CID during activation by the PCD

H.2.9.1 Scope

The purpose of this test is to determine the handling of the CID according to ISO/IEC 14443-4:2018, 5.6.3. This test shall be executed for at least CID set to 0, 1 and 14 if the CID can be chosen by the UT. Else, only the CID chosen by the PCD shall be used.

H.2.9.2 Apparatus

See H.1.

H.2.9.3 Procedure

Use the sequence a) to c) to put the PCD into the state required by this test.

a) Place the LT into the PCD operating volume and record the presence and the content of the PCD commands.

b) The UT performs the activation procedure according to H.1.9.1.

c) The LT answers relevant anticollision messages and waits until the PCD sends the RATS. The LT answers with ATS.

For each test from the scenario described in Table H.22, when supported by the PCD, use the sequence d) to h).

d) Put the PCD into the state required by this test.

e) The LT waits until the PCD applies the command as described in PCD column.

f) The LT answers as described in the LT column.

g) The UT sends the SEND_UT_APDU(UT_TEST_COMMAND1) to the PCD.

h) The PCD is expected to send I-block to the LT applying the condition as described in the PCD column.

Table H.22 — Scenario H.12: Handling of the CID

Test	PCD		LT
CID = n not equal to 0 and receive CID is supported	RATS (CID not equal 0)	→	
		←	ATS (CID supported)
	Any valid command using CID	→	
CID = n not equal to 0 and receive CID is not supported	RATS (CID not equal 0)	→	
		←	ATS (CID not supported)
	Any valid command without CID	→	
CID = n equal to 0 and receive CID is supported	RATS (CID equal to 0)	→	
		←	ATS (CID supported)
	Any valid command using CID = 0 or without CID	→	
CID = n equal to 0 and receive CID is not supported	RATS (CID equal to 0)	→	
		←	ATS (CID not supported)
	Any valid command without CID	→	

H.2.9.4 Test report

Fill the appropriate row in Table H.58 according to Table H.23.

Table H.23 — Result criteria for handling of the CID during activation by the PCD

Explanation	Test result
Only when the PCD's behavior matches the expected scenario	PASS
Any other case	FAIL

H.2.10 Handling of parity bit

H.2.10.1 Scope

The purpose of this test is to determine the behavior of a PCD in Type A when receiving PICC messages with parity error according to ISO/IEC 14443-3:2018, 6.2.3.2.

H.2.10.2 Apparatus

See H.1.

H.2.10.3 Procedure

Place the LT into the PCD operating volume.

During the following procedure the RF Input/Receive data shall be continuously monitored and verified correct to ISO/IEC 14443-2. All signal transitions (level and timing) as well as the logical content of the communication shall be recorded.

Use the following sequence:

a) The UT performs the protocol activation procedure according to H.1.9.2. Interface byte TA(1) is set to (00000000)b (only bit rate of $f_c/128$ in both directions).

b) The UT sends the SEND_UT_APDU (UT_TEST_COMMAND1) to the PCD.

c) The LT waits until the PCD sends an I-block to the LT with the INF field containing the UT_TEST_COMMAND1.

d) The LT sends an erroneous I-block to the PCD with a wrong parity bit.

e) The PCD shall send $R(NAK)_0$.

f) The LT sends again the erroneous block to the PCD.

g) The PCD shall send either $R(NAK)_0$ or S(DESELECT) request.

Table H.24 gives part of the procedure as a scenario.

Table H.24 — Scenario H.31: Parity bit error in I-block

PCD		LT
$I(0)_0$(INF = UT_TEST_COMMAND1)	→	
	←	$I(0)_0$(INF = answer to UT_TEST_COMMAND1, wrong parity bit)[a]
$R(NAK)_0$ ('BA' CID CRC or 'B2' CRC)	→	
	←	$I(0)_0$(INF = answer to UT_TEST_COMMAND1, wrong parity bit)[a]
$R(NAK)_0$ ('BA' CID CRC or 'B2' CRC) or S(DESELECT)	→	

[a] The parity error is simulated on the first transmitted byte of the frame by reversing the parity bit.

NOTE There is no test at PICC to PCD bit rates higher than $f_c/128$ as the inverted parity bit of the last byte is considered as End of Communication.

H.2.10.4 Test report

Fill the appropriate row in Table H.58 according to Table H.25.

Table H.25 — Result criteria for handling of parity errors

Explanation	Test result
Only when the PCD's behavior matches the expected scenario	PASS
Any other case	FAIL

H.3 Type B specific test methods

H.3.1 Void

I/O transmission timing, which was defined in former editions of this document, is no longer present.

H.3.2 Frame size selection mechanism

H.3.2.1 Scope

The purpose of this test is to analyze the frame size selection mechanism according to ISO/IEC 14443-3:2018, 7.9.

This test shall be executed for at least Maximum Frame Size Code in ATQB set to '0', '1', '8' and 'C'.

H.3.2.2 Apparatus

See H.1.

H.3.2.3 Procedure

Place the LT into the PCD operating volume and record the presence and the content of the PCD commands.

For each of the Maximum Frame Size Code in ATQB parameters, run the following sequence:

a) The UT performs the activation procedure according to H.1.9.1.

b) The LT waits until the PCD sends a valid REQB/WUPB command frame.

c) The LT answers with ATQB. For the purpose of this test the LT returns in the second protocol info byte (see ISO/IEC 14443-3:2018, 7.9.4) the Maximum Frame Size Code in ATQB parameter and indicates compliancy with ISO/IEC 14443-4.

d) The PCD shall send a valid ATTRIB command frame.

e) The LT sends Answer to ATTRIB command.

f) The PCD shall return the result code, in accordance with Table H.1, of its action to the UT.

g) The UT sends the SEND_UT_APDU(UT_TEST_COMMAND2) to the PCD.

h) The PCD shall send an $I(1)_0$ block with its length not exceeding the maximum frame size resulting from the Maximum Frame Size Code in ATQB indicated by the PICC, as shown in the PCD column of Scenario H.13

Table H.26 gives part of the procedure as a scenario.

Table H.26 — Scenario H.13: Frame size selection mechanism

PCD		LT
REQB/WUPB	→	
	←	ATQB
ATTRIB command frame	→	
	←	Answer to ATTRIB command
$I(1)_0$(INF = the first chain includes the first block of UT_TEST_COMMAND2), with its length not exceeding the maximum frame size of the LT.	→	

H.3.2.4 Test report

Fill the appropriate row in Table H.60 according to Table H.27.

Table H.27 — Result criteria for frame size selection mechanism

Explanation	Test result
Only when the PCD's behavior matches the expected scenario for all tested values of Maximum Frame Size Code in ATQB values	PASS
Any other case	FAIL

H.3.3 Handling of the CID during activation by the PCD

H.3.3.1 Scope

The purpose of this test is to determine the handling of the CID according to ISO/IEC 14443-3.

This test shall be executed for at least CID set to 0, 1 and 14 if the CID can be chosen by the UT. Else, only the CID chosen by the PCD shall be used.

H.3.3.2 Apparatus

See H.1.

H.3.3.3 Procedure

H.3.3.3.1 General

Place the LT into the PCD operating volume and record the presence and the content of the PCD commands.

H.3.3.3.2 Procedure 1

Use the following sequence:

a) The UT performs the activation procedure according to H.1.9.1.

b) The LT waits until the PCD sends the REQB/WUPB command.

c) The LT sends ATQB with Frame Option bits (b2,b1) equal (00)b. It means: CID and NAD are not supported.

d) The LT waits until the PCD sends the ATTRIB command. The PCD shall send a valid ATTRIB command frame with Param 4 byte equals 0.

e) The LT sends Answer to ATTRIB command with CID value equals 0.

f) The PCD shall return the result code, in accordance with Table H.1, of its action to the UT.

g) The UT sends the SEND_UT_APDU(UT_TEST_COMMAND1) to the PCD.

h) The PCD shall send the following I(0 or 1)$_0$ block without NAD and CID.

i) The LT remains MUTE.

j) The PCD shall send at least one R(NAK)$_0$ without NAD and CID.

k) The LT remains MUTE.

l) The PCD shall send S(DESELECT) request without NAD and CID.

Table H.28 gives part of the procedure as a scenario.

Table H.28 — Scenario H.14: Handling of the CID, Procedure 1

PCD		LT
REQB/WUPB	→	
	←	ATQB Frame Option Bits (b2,b1) = (00)b
ATTRIB command frame (e.g.'1D 12 23 34 45 00 05 01 00' CRC_B)	→	
	←	Answer to ATTRIB command
I(0 or 1)$_0$ without NAD and CID	→	
	←	Mute
R(NAK)$_0$ without NAD and CID	→	
	←	Mute
S(DESELECT) request without NAD and CID	→	

H.3.3.3.3 Procedure 2

Use the following sequence:

a) The UT performs the activation procedure according to H.1.9.1.

b) The LT waits until the PCD sends the REQB/WUPB command.

c) The LT sends ATQB with Frame Option bits (b2,b1) equal (01)b. It means: CID is supported and NAD is not supported.

d) The LT waits until the PCD sends the ATTRIB command. The PCD shall send a valid ATTRIB command frame, using Param 4 byte equals '0X' (CID = X in the range from 0 to 14).

e) The LT sends Answer to ATTRIB command with CID value assigned by the PCD on step d) in Param 4 byte.

f) The PCD shall return the result code, in accordance with Table H.1, of its action to the UT.

g) The UT sends the SEND_UT_APDU(UT_TEST_COMMAND1) to the PCD.

h) The PCD shall send the following I(0 or 1)$_0$ block using CID value assigned by the PCD on step d) or, optionally, using no CID if CID = 0. The PCD shall not use NAD in this I(0 or 1)$_0$ block.

i) The LT remains MUTE.

j) The PCD shall send at least one $R(NAK)_0$ with CID value equals to '0X' and without NAD.

k) The LT remains MUTE.

k) The PCD shall send S(DESELECT) request with CID value equals to '0X' and without NAD.

Table H.29 gives part of the procedure as a scenario.

Table H.29 — Scenario H.15: Handling of the CID, Procedure 2

PCD		LT
REQB/WUPB	→	
	←	ATQB Frame Option Bits (b2,b1) = (01)b
ATTRIB command frame with Param 4 = '0X' (e.g.'1D 12 23 34 45 00 05 01 0X' CRC_B), 'X' in the range of 0 to 14	→	
	←	Answer to ATTRIB command
$I(0 \text{ or } 1)_0$ with CID value equal to '0X' and without NAD	→	
	←	Mute
$R(NAK)_0$ with CID value equals '0X' and without NAD	→	
	←	Mute
S(DESELECT) request with CID value equals '0X' and without NAD	→	

H.3.3.4 Test report

Fill the appropriate row in Table H.60 according to Table H.30.

Table H.30 — Report criteria for handling of the CID during activation by the PCD

Explanation	Test result
Only when the PCD's behavior matches each procedure expected scenario	PASS
Any other case	FAIL

H.3.4 Frame Delay Time PICC to PCD (TR2)

H.3.4.1 Scope

The purpose of this test is to determine the timing between a PICC frame and the next PCD frame according to the minimum TR2 coding as defined in ISO/IEC 14443-3:2018, 7.9.4.4.

This test shall be executed for all the minimum TR2 values:

1) b3 and b2 of Protocol_Type equal to (00)b;

2) b3 and b2 of Protocol_Type equal to (01)b;

3) b3 and b2 of Protocol_Type equal to (10)b;

4) b3 and b2 of Protocol_Type equal to (11)b.

H.3.4.2 Apparatus

See H.1.

H.3.4.3 Procedure

Place the LT into the PCD operating volume.

During the following procedure the RF Input/Receive data shall be continuously monitored and verified correct to ISO/IEC 14443-2. All signal transitions (level and timing) as well as the logical content of the communication shall be recorded.

Use the following sequence:

a) The UT performs the activation procedure according to H.1.9.1.

b) The LT waits until the PCD sends a valid REQB/WUPB command frame.

c) The LT answers with a valid ATQB coding the minimum TR2 as expected.

d) The PCD shall send a valid ATTRIB command frame.

e) The LT sends Answer to ATTRIB.

f) The PCD shall return the result code, in accordance with Table H.1, of its action to the UT.

g) The UT sends the SEND_UT_APDU(UT_TEST_COMMAND1) to the PCD.

h) The LT waits until the PCD sends an I-block to the LT with the INF field containing the UT_TEST_COMMAND1.

i) The LT sends an I-block (containing some response UT_APDU with answer to the UT_TEST_COMMAND1) to the PCD.

j) The PCD is expected to transfer the response UT_APDU back to the UT. Check at the UT that this response UT_APDU block is correctly accepted.

k) The UT sends the SEND_UT_APDU(UT_TEST_COMMAND1) to the PCD.

l) The LT sends S(WTX) request using any valid WTXM.

m) The PCD shall send S(WTX) response using the same WTXM as in the S(WTX) request. If it does not meet the expected response, end the test at this point.

n) The LT sends an I-block (containing some response UT_APDU with answer to the UT_TEST_COMMAND1) to the PCD.

o) Measure the time delay between the PICC EOF start and PCD SOF start (see ISO/IEC 14443-3:2018, 7.1.7) for each command.

Table H.31 gives part of the procedure as a scenario.

Table H.31 — Scenario H.27: Frame Delay Time PICC to PCD (TR2)

PCD		LT
REQB/WUPB	→	
	←	ATQB (coding the minimum TR2 as expected)
ATTRIB command frame	→	
	←	Answer to ATTRIB
$I(0)_0$(INF = UT_TEST_COMMAND1)	→	
	←	$I(0)_0$(INF = answer to UT_TEST_COMMAND1)
NOTE If any of the steps requires command sent by the UT, the subsequent steps may be skipped.		

Table H.31 *(continued)*

PCD		LT
$I(0)_1$(INF = UT_TEST_COMMAND1)	→	
	←	S(WTX) request
S(WTX) response	→	
	←	$I(0)_1$(INF = answer to UT_TEST_COMMAND1)
NOTE If any of the steps requires command sent by the UT, the subsequent steps may be skipped.		

H.3.4.4 Test report

Report the signal recording. Fill item 6 of Table H.57 with minimum measured value of Frame Delay Time and the appropriate row of Table H.59 in accordance with Table H.32.

Table H.32 — Report criteria for Frame Delay Time PICC to PCD (TR2)

Explanation	Test result
Only when the PCD's Frame Delay Time PICC to PCD (TR2) always exceed the minimum value defined in ISO/IEC 14443-3:2018, 7.9.4.4	PASS
Any other case	FAIL

H.3.5 Handling of Start-up Frame Guard Time

H.3.5.1 Scope

The purpose of this test is to determine the PCD transmission timing according to ISO/IEC 14443-3:2018, 7.9.4.7. This test shall be executed for at least SFGI set to 0, 1 and 14.

This test is applicable only if PCD supports extended ATQB as described in ISO/IEC 14443-3:2018, 7.9.4.7.

H.3.5.2 Apparatus

See H.1.

H.3.5.3 Procedure

Place the LT into the PCD operating volume and record the presence and the content of the PCD commands.

During the following procedure the RF Input/Receive data shall be continuously monitored and verified correct to ISO/IEC 14443-2. All signal transitions (level and timing) as well as the logical content of the communication shall be recorded.

Use the following sequence:

a) The UT performs the activation procedure according to H.1.9.1.

b) The LT waits until the PCD sends a valid REQB/WUPB command frame (b5 set to (1)b).

c) The LT answers with a valid extended ATQB coding the SFGI as expected.

d) The PCD shall send a valid ATTRIB command frame.

e) The LT answers with a valid Answer to ATTRIB.

f) The UT sends the SEND_UT_APDU (UT_TEST_COMMAND1) to the PCD.

g) The PCD shall send the next block after a minimum delay of SFGT.

Table H.33 gives part of the procedure as a scenario.

Table H.33 — Scenario H.33: Handling of Start-up Frame Guard Time

PCD		LT
REQB/WUPB supporting Extended ATQB	→	
	←	Extended ATQB (coding the SFGI as expected)
ATTRIB command frame	→	
	←	Answer to ATTRIB
Next command (sent not earlier than SFGT)	→	

H.3.5.4 Test report

Report the signal recording. Fill item 7 of Table H.57 with measured values and the appropriate row of Table H.59 in accordance with Table H.34.

Table H.34 — Result criteria for handling of Start-up Frame Guard Time

Explanation	Test result
Only when the PCD's behavior matches the expected scenario for all tested SFGI values	PASS
Any other case	FAIL

H.3.6 Type B PCD framing tests

H.3.6.1 Scope

The purpose of this test is to determine the behavior of a PCD in Type B when receiving PICC messages according to ISO/IEC 14443-3:2018, 7.1.

H.3.6.2 Apparatus

See H.1.

H.3.6.3 Procedure

This procedure shall be repeated for combinations of PCD to PICC and PICC to PCD bit rates supported and selected by the PCD, so that all supported bit rates in each direction are used at least once during the tests.

Place the LT into the PCD operating volume and record the presence and the content of the PCD commands.

Perform the following steps for each row of Table H.35:

Use the following sequence:

a) The UT performs the activation procedure according to H.1.9.1.

b) The LT waits until the PCD sends a valid REQB/WUPB command frame.

c) The LT answers with a valid ATQB after setting the frame parameters of the LT according to Table H.35.

d) The PCD shall send a valid ATTRIB command frame.

e) The LT sends Answer to ATTRIB.

f) The PCD shall return the answer to ATTRIB, in accordance with Table H.1, of its action to the UT.

g) The PCD may send an S(PARAMETERS) block to send Bit rates Request, possibly after other S(PARAMETERS) exchanges. If not, skip steps h) to j).

h) The LT answers with an S(PARAMETERS) block that includes the framing options provided in Table H.18 and an appropriate combination of bit rates.

i) The PCD may send an S(PARAMETERS) activating bit rates and framing options. If not, skip step j).

j) The LT acknowledges the received S(PARAMETERS) block with an S(PARAMETERS) block response.

k) The UT sends the SEND_UT_APDU(UT_TEST_COMMAND1) to the PCD.

l) The LT waits until the PCD sends an I-block to the LT with the INF field containing the UT_TEST_COMMAND1.

m) The LT sends I-block (containing some response UT_APDU with answer to the UT_TEST_COMMAND1) to the PCD.

n) The PCD is expected to transfer the response UT_APDU back to the UT. Check at the UT that this response UT_APDU block is correctly accepted.

Table H.35 — Type B PICC to PCD frame parameters

Test No.	**EGT** [etu]	**SOF low** [etu]	**SOF high** [etu]	**EOF** [etu]	**f_s to OFF** [etu]	**TR0**	**TR1**	**Framing options (b2b1)**[a]
1	0	10,5	2,5	10,5	1	$1600/f_c$	$140/f_s$	(00)b
2	0	No SOF if "No SOF required" in ATTRIB from Answer to ATTRIB and following[b]	No SOF if "No SOF required" in ATTRIB from Answer to ATTRIB and following[b]	No EOF if "No EOF required" in ATTRIB from Answer to ATTRIB and following[b]	1	$1600/f_c$	$140/f_s$	(00)b
3	0	$10 - 1/f_s$ for $f_c/128$ $10 - 0{,}5/f_s$ for $f_c/64$ 10 for other	$2 - 1/f_s$ for $f_c/128$ $2 - 0{,}5/f_s$ for $f_c/64$ 2 for other	$10 - 1/f_s$ for $f_c/128$ $10 - 0{,}5/f_s$ for $f_c/64$ 10 for other	0	$1600/f_c$	$140/f_s$	(00)b
4	2,125	$10 - 1/f_s$ for $f_c/128$ $10 - 0{,}5/f_s$ for $f_c/64$ 10 for other	$2 - 1/f_s$ for $f_c/128$ $2 - 0{,}5/f_s$ for $f_c/64$ 2 for other	$10 - 1/f_s$ for $f_c/128$ $10 - 0{,}5/f_s$ for $f_c/64$ 10 for other	$2 + 1/f_s$	$1600/f_c$	$140/f_s$	(00)b

[a] Only applicable if selected by the PCD.

[b] Only applicable at a bit rate of $f_c/128$.

[c] min TR0 is defined in ISO/IEC 14443-2:2020, 9.2.5 for ATQB and in ISO/IEC 14443-3:2018, 7.10.3.1 for other frames.

[d] min TR1 is defined in ISO/IEC 14443-2:2020, 9.2.5 for ATQB and in ISO/IEC 14443-3:2018, 7.10.3.2 for other frames.

Table H.35 *(continued)*

Test No.	EGT [etu]	SOF low [etu]	SOF high [etu]	EOF [etu]	f_s to OFF [etu]	TR0	TR1	Framing options (b2b1)[a]
5	0	$11 + 1/f_s$ for $f_c/128$ $11 + 0{,}5/f_s$ for $f_c/64$ 11 for other	$2 - 1/f_s$ for $f_c/128$ $2 - 0{,}5/f_s$ for $f_c/64$ 2 for other	$10 - 1/f_s$ for $f_c/128$ $10 - 0{,}5/f_s$ for $f_c/64$ 10 for other	0	$1600/f_c$	$140/f_s$	(00)b
6	0	$10 - 1/f_s$ for $f_c/128$ $10 - 0{,}5/f_s$ for $f_c/64$ 10 for other	$3 + 1/f_s$ for $f_c/128$ $3 + 0{,}5/f_s$ for $f_c/64$ 3 for other	$10 - 1/f_s$ for $f_c/128$ $10 - 0{,}5/f_s$ for $f_c/64$ 10 for other	0	$1600/f_c$	$140/f_s$	(00)b
7	2,125	$11 + 1/f_s$ for $f_c/128$ $11 + 0{,}5/f_s$ for $f_c/64$ 11 for other	$3 + 1/f_s$ for $f_c/128$ $3 + 0{,}5/f_s$ for $f_c/64$ 3 for other	$11 + 1/f_s$ for $f_c/128$ $11 + 0{,}5/f_s$ for $f_c/64$ 11 for other	$2 + 1/f_s$	$1600/f_c$	$140/f_s$	(00)b
8	0	$10 - 1/f_s$ for $f_c/128$ $10 - 0{,}5/f_s$ for $f_c/64$ 10 for other	$2 - 1/f_s$ for $f_c/128$ $2 - 0{,}5/f_s$ for $f_c/64$ 2 for other	$10 - 1/f_s$ for $f_c/128$ $10 - 0{,}5/f_s$ for $f_c/64$ 10 for other	0	min TR0 $- 16/f_c$[c]	min TR1 $- 1/f_s$[d]	(00)b
9	2,125	$11 + 1/f_s$ for $f_c/128$ $11 + 0{,}5/f_s$ for $f_c/64$ 11 for other	$3 + 1/f_s$ for $f_c/128$ $3 + 0{,}5/f_s$ for $f_c/64$ 3 for other	$11 + 1/f_s$ for $f_c/128$ $11 + 0{,}5/f_s$ for $f_c/64$ 11 for other	$2 + 1/f_s$	$(256/f_s) \times 2^{FWI} - TR1 + 16/f_c$	$200/f_s$	(00)b
10	2,125	$11 + 1/f_s$ for $f_c/128$ $11 + 0{,}5/f_s$ for $f_c/64$ 11 for other	$3 + 1/f_s$ for $f_c/128$ $3 + 0{,}5/f_s$ for $f_c/64$ 3 for other	$11 + 1/f_s$ for $f_c/128$ $11 + 0{,}5/f_s$ for $f_c/64$ 11 for other	$2 + 1/f_s$	$(256/f_s) \times 2^{FWI} - TR1 + 16/f_c$	min TR1 $- 1/f_s$[d]	(00)b
11	0	$10 - 1/f_s$ for $f_c/128$ $10 - 0{,}5/f_s$ for $f_c/64$ 10 for other	$2 - 1/f_s$ for $f_c/128$ $2 - 0{,}5/f_s$ for $f_c/64$ 2 for other	$10 - 1/f_s$ for $f_c/128$ $10 - 0{,}5/f_s$ for $f_c/64$ 10 for other	0	min TR0 $- 16/f_c$[c]	$200/f_s$	(00)b

[a] Only applicable if selected by the PCD.

[b] Only applicable at a bit rate of $f_c/128$.

[c] min TR0 is defined in ISO/IEC 14443-2:2020, 9.2.5 for ATQB and in ISO/IEC 14443-3:2018, 7.10.3.1 for other frames.

[d] min TR1 is defined in ISO/IEC 14443-2:2020, 9.2.5 for ATQB and in ISO/IEC 14443-3:2018, 7.10.3.2 for other frames.

Table H.35 *(continued)*

Test No.	EGT [etu]	SOF low [etu]	SOF high [etu]	EOF [etu]	f_s to OFF [etu]	TR0	TR1	Framing options (b2b1)[a]
12	0	No SOF if requested in framing options with S-block	No SOF if requested in framing options with S-block	No EOF if requested in framing options with S-block	0	min TR0 – $16/f_c$[c]	min TR1 – $1/f_s$[d]	(10)b
13	0	$10 - 1/f_s$ for $f_c/128$ $10 - 0{,}5/f_s$ for $f_c/64$ 10 for other	$2 - 1/f_s$ for $f_c/128$ $2 - 0{,}5/f_s$ for $f_c/64$ 2 for other	$10 - 1/f_s$ for $f_c/128$ $10 - 0{,}5/f_s$ for $f_c/64$ 10 for other	0	min TR0 – $16/f_c$[c]	min TR1 – $1/f_s$[d]	(01)b

[a] Only applicable if selected by the PCD.

[b] Only applicable at a bit rate of $f_c/128$.

[c] min TR0 is defined in ISO/IEC 14443-2:2020, 9.2.5 for ATQB and in ISO/IEC 14443-3:2018, 7.10.3.1 for other frames.

[d] min TR1 is defined in ISO/IEC 14443-2:2020, 9.2.5 for ATQB and in ISO/IEC 14443-3:2018, 7.10.3.2 for other frames.

H.3.6.4 Test report

Fill the appropriate row in Table H.60 according to Table H.36.

Table H.36 — Result criteria for PCD reception test

Explanation	Test result
Only when the PCD's behavior matches the expected scenario for all tested frame parameters	PASS
Any other case	FAIL

H.4 Test method for logical operations of the PCD

H.4.1 General

All test methods described in this subclause, except H.4.2, shall be applied twice, once for Type A signal interface and once for Type B signal interface.

H.4.2 Handling of the polling loop

H.4.2.1 Scope

The purpose of this test is to determine the behavior of the PCD during polling. The conditions defined in ISO/IEC 14443-3:2018, 5.2.1 shall apply.

H.4.2.2 Apparatus

See H.1.

H.4.2.3 Procedure

Place the LT into the PCD operating volume and record the presence and the content of the PCD commands.

During the following procedure the RF Input/Receive data shall be continuously monitored and verified correct to ISO/IEC 14443-2. All signal transitions (level and timing) as well as the logical content of the communication shall be recorded.

Use the following sequence:

a) The UT performs the protocol activation procedure according to H.1.9.1.

b) The LT waits until the PCD sends a valid REQA/WUPA command frame and a valid REQB/WUPB command frame, in any order and repetition.

H.4.2.4 Test report

Fill the appropriate row in Table H.60 considering results for both for Type A and Type B according to Table H.37.

Table H.37 — Result criteria for handling of the polling loop

Explanation	Test result
Only when the PCD sends at least one REQA/WUPA command frame (with at least 5 ms of unmodulated operating field before it) and at least one REQB/WUPB command frame (with at least 5 ms of unmodulated operating field before it).	PASS
Any other case	FAIL

H.4.3 Reaction of the PCD to request for waiting time extension

H.4.3.1 Scope

The purpose of this test is to determine the behavior of the PCD when the PICC uses a request for a waiting time extension (see ISO/IEC 14443-4:2018, 7.3). The mechanism of maintenance of WTX by the PCD is also tested.

This test shall be executed for at least FWI set to 0, 1 and 14 with TR0 and TR1 set to designate the default value in the LT, if it emulates a Type B PICC.

This test combination shall be executed for at least WTXM set to 1, 3 and 59 according to Table H.38.

Table H.38 — Minimum combinations

Combination	FWI	WTXM
1	0	1
2	0	3
3	0	59
4	1	1
5	1	3
6	1	59
7	14	1
8	14	3
9	14	59

H.4.3.2 Apparatus

See H.1.

H.4.3.3 Procedure

Place the LT into the PCD operating volume and record the presence and the content of the PCD commands.

During the following procedure the RF Input/Receive data shall be continuously monitored and verified correct to ISO/IEC 14443-2. All signal transitions (level and timing) as well as the logical content of the communication shall be recorded.

The UT performs the protocol activation procedure according to H.1.9.2 for Type A or H.1.9.3 for Type B.

Use the following sequence:

a) The UT sends the SEND_UT_APDU(UT_TEST_COMMAND1) to the PCD.

b) The LT waits until the PCD sends an I-block to the LT with the INF field containing the UT_TEST_COMMAND1.

c) Set the following bit timing parameters at the LT:

   1) Frame waiting time (FWT): $(256 \times 16/f_c) \times 2^{FWI}$ (see ISO/IEC 14443-4:2018, 7.2)

   2) Extra Guard Time, Type B (EGT): The maximum value the PCD shall accept (see ISO/IEC 14443-3:2018, 7.1.2)

d) Just before FWT expiration, the LT sends S(WTX) request.

e) The PCD shall send S(WTX) response with INF(b6 to b1) = WTXM. If it does not meet the expected response, end the test at this point.

f) Set the following bit timing parameters at the LT:

   1) Temporary frame waiting time (FWT_{TEMP}): WTXM × $(256 \times 16/f_c) \times 2^{FWI}$ (see ISO/IEC 14443-4:2018, 7.2 and 7.3)

   2) Extra Guard Time, Type B (EGT): The maximum value the PCD shall accept (see ISO/IEC 14443-3:2018, 7.1.2)

g) Just before FWT_{TEMP} expiration, the LT sends the answer to the command UT_TEST_COMMAND1 sent in b).

h) The PCD is expected to transfer the response UT_APDU (answer to the command UT_TEST_COMMAND1) back to the UT.

i) The UT sends the SEND_UT_APDU(UT_TEST_COMMAND1) to the PCD.

j) The LT waits until the PCD sends an I-block to the LT with the INF field containing the UT_TEST_COMMAND1. The PCD shall reset the FWT at this point.

k) The LT remains Mute for at least the expected FWT. This fact means FWT timeout for the PCD.

l) Record the presence, the content and timing of the PCD-response. The PCD shall send an R(NAK) block only after FWT expires (ISO/IEC 14443-4:2018, 7.5.5.2).

m) Measure and record the time between the end of the PCD frame from step i) and start of PCD frame from step k).

Scenario H.16: Void

Scenario H.16, which was defined in former editions of this document, is no longer present.

Table H.39 gives part of the procedure as a scenario.

Table H.39 — Scenario H.17: PCD reaction to the LT waiting time extension request

PCD		LT
$I(0)_0$(INF = UT_TEST_COMMAND1)	→	
	←	S(WTX) request (sent just before FWT_{TEMP} expiration) (WTXM belongs to the test set)
S(WTX) response	→	
	←	$I(0)_0$(INF = answer to UT_TEST_COMMAND1) (sent just before FWT_{TEMP} expiration)
$I(0)_1$(INF = UT_TEST_COMMAND1)	→	
	←	Mute
R(NAK)	→	

H.4.3.4 Test report

The PCD's behavior shall match the expected Scenario H.17 for all FWI and WTXM values defined in H.4.3 in all combinations tested. Only when each test with these different values is passed, the test result is PASS. In any other case, the test result is FAIL.

Fill the appropriate row in Table H.60 for both Type A and Type B according to Table H.40.

Table H.40 — Result criteria for reaction of the PCD to request for waiting time extension

Explanation	Test result
Only when the answer to TEST_COMMAND_1 was not sent back to the UT or when the PCD sent the R(NAK) before FWT expired for at least one FWI one WTXM values defined in H.4.3 (the minimum combinations to test are given in Table H.21)	FAIL
Any other case	PASS

H.4.4 Error detection and recovery

H.4.4.1 Scope

The purpose of this test is to determine the behavior of PCD when an error occurs according to ISO/IEC 14443-4:2018, 7.5.7.

NOTE In this subclause and the following subclauses, "erroneous block" is a frame with a wrong CRC.

H.4.4.2 Apparatus

See H.1.

H.4.4.3 Procedure

H.4.4.3.1 General

Place the LT into the PCD operating volume and record the presence and the content of the PCD commands.

During the following procedure the RF Input/Receive data shall be continuously monitored and verified correct to ISO/IEC 14443-2. All signal transitions (level and timing) as well as the logical content of the communication shall be recorded.

The UT performs the protocol activation procedure according to H.1.9.2 for Type A or H.1.9.3 for Type B.

H.4.4.3.2 Procedure 1 (ISO/IEC 14443-4:2018, Annex B, Scenario 12)

Use the following sequence immediately after procedure H.4.4.3.1:

a) The UT sends the SEND_UT_APDU(UT_TEST_COMMAND1) to the PCD.

b) The LT waits until the PCD sends an I-block to the LT with the INF field containing the UT_TEST_COMMAND1.

c) The LT sends an erroneous I-block to the PCD.

d) The PCD shall send $R(NAK)_0$.

e) The LT sends I-block (containing some response UT_APDU with answer to the UT_TEST_COMMAND1) to the PCD.

f) The PCD is expected to transfer the response UT_APDU back to the UT. Check at the UT that this response UT_APDU block is correctly accepted.

Table H.41 gives part of the procedure as a scenario.

Table H.41 — Scenario H.18: Error detection and recovery of a transmission error by the PCD, Procedure 1

PCD		LT
$I(0)_0$(INF = UT_TEST_COMMAND1)	→	
	←	$I(0)_0$(INF = answer to UT_TEST_COMMAND1, wrong CRC)
$R(NAK)_0$ ('BA' CID CRC or 'B2' CRC)	→	
	←	$I(0)_0$(INF = answer to UT_TEST_COMMAND1)

H.4.4.3.3 Procedure 2 (ISO/IEC 14443-4:2018, 7.5.5.2 rule 4)

Use the following sequence immediately after procedure H.4.4.3.1:

a) The UT sends the SEND_UT_APDU(UT_TEST_COMMAND1) to the PCD.

b) The LT waits until the PCD sends an I-block to the LT with the INF field containing the UT_TEST_COMMAND1.

c) The LT sends an erroneous block to the PCD.

d) The PCD shall send $R(NAK)_0$.

e) The LT sends a second erroneous block to the PCD.

f) The PCD shall send either $R(NAK)_0$ or S(DESELECT) request.

Table H.42 gives part of the procedure as a scenario.

Table H.42 — Scenario H.19: Error detection and recovery of a transmission error by the PCD, Procedure 2

PCD		LT
$I(0)_0$(INF = UT_TEST_COMMAND1)	→	
	←	$I(0)_0$(INF = answer to UT_TEST_COMMAND1, wrong CRC)
$R(NAK)_0$ ('BA' CID CRC or 'B2' CRC)	→	
	←	$I(0)_0$(INF = answer to UT_TEST_COMMAND1, wrong CRC)
$R(NAK)_0$ ('BA' CID CRC or 'B2' CRC) or S(DESELECT)	→	

H.4.4.3.4 Procedure 3 (ISO/IEC 14443-4:2018, 7.5.5.2 rule 4)

Use the following sequence immediately after procedure H.4.4.3.1:

a) The UT sends the SEND_UT_APDU(UT_TEST_COMMAND1) to the PCD.

b) The LT waits until the PCD sends an I-block to the LT with the INF field containing the UT_TEST_COMMAND1.

c) Maintain the LT Mute.

d) Record all responses from the PCD. The PCD shall send $R(NAK)_0$ at least once.

Table H.43 gives part of the procedure as a scenario.

Table H.43 — Scenario H.20: Error detection and recovery of a transmission error by the PCD, Procedure 3

PCD		LT
$I(0)_0$(INF = UT_TEST_COMMAND1)	→	
	←	Mute
$R(NAK)_0$ ('BA' CID CRC or 'B2' CRC)	→	
	←	Mute
$R(NAK)_0$ ('BA' CID CRC or 'B2' CRC) or S(DESELECT)	→	

H.4.4.3.5 Procedure 4 (ISO/IEC 14443-4:2018, 7.5.5.2 rule 7)

Use the following sequence immediately after procedure H.4.4.3.1:

a) The UT sends the SEND_UT_APDU(UT_TEST_COMMAND1) to the PCD.

b) The LT waits until the PCD sends an I-block to the LT with the INF field containing the UT_TEST_COMMAND1.

c) The LT sends $R(ACK)_0$ (because of infringement of protocol rules by the LT).

d) The PCD shall send a S(DESELECT) request.

Table H.44 gives part of the procedure as a scenario.

Table H.44 — Scenario H.21: Error detection and recovery of a transmission error by the PCD, Procedure 4

PCD		LT
$I(0)_0$(INF = UT_TEST_COMMAND1)	→	
	←	$R(ACK)_0$ ('AA' CID CRC or 'A2' CRC)[a]
S(DESELECT) request	→	

[a] If the PCD block contains a CID, the left option shall be used, else the right option shall be used.

H.4.4.3.6 Procedure 5 (with chaining) (ISO/IEC 14443-4:2018, 7.5.5.2 rule 5, ISO/IEC 14443-4:2018, Annex B, Scenario 23)

Use the following sequence immediately after procedure H.4.4.3.1:

a) The UT sends the SEND_UT_APDU(UT_TEST_COMMAND1) to the PCD.

b) The LT waits until the PCD sends an I-block to the LT, with the INF field containing the UT_TEST_COMMAND1.

c) The LT sends the first block $I(1)_0$ of the chain and waits for the PCD response.

d) The PCD shall send $R(ACK)_1$.

e) The LT sends an erroneous block $I(0)_1$ to the PCD.

f) The PCD shall send $R(ACK)_1$.

Table H.45 gives part of the procedure as a scenario.

Table H.45 — Scenario H.22: Error detection and recovery of a transmission error by the PCD, Procedure 5

PCD		LT
$I(0)_0$(INF = UT_TEST_COMMAND1)	→	
	←	$I(1)_0$(INF = the first chain of the answer to UT_TEST_COMMAND1)
$R(ACK)_1$ ('AB' CID CRC or 'A3' CRC)	→	
	←	$I(0)_1$(INF = the last chain of the answer to UT_TEST_COMMAND1, wrong CRC)
$R(ACK)_1$	→	

H.4.4.3.7 Procedure 6 (ISO/IEC 14443-4:2018, 7.5.5.2)

Use the following sequence immediately after procedure H.4.4.3.1:

a) The UT sends the SEND_UT_APDU(UT_TEST_COMMAND1) to the PCD.

b) The LT waits until the PCD sends an I-block to the LT with the INF field containing the UT_TEST_COMMAND1.

c) The LT sends an erroneous block to the PCD.

d) The PCD shall send $R(NAK)_0$.

e) The LT remains Mute.

f) Record all responses from the PCD. The PCD shall send either $R(NAK)_0$ or S(DESELECT) request.

Table H.46 gives part of the procedure as a scenario.

Table H.46 — Scenario H.23: Error detection and recovery of a transmission error by the PCD, Procedure 6

PCD		LT
$I(0)_0$(INF = UT_TEST_COMMAND1)	→	
	←	$I(0)_0$(INF = answer to UT_TEST_COMMAND1, wrong CRC)
$R(NAK)_0$ ('BA' CID CRC or 'B2' CRC)	→	
	←	Mute
$R(NAK)_0$ or S(DESELECT) request	→	

H.4.4.3.8 Procedure 7 (ISO/IEC 14443-4:2018, 7.5.7, Annex B, Scenario 14)

Use the following sequence immediately after procedure H.4.4.3.1:

a) The UT sends the SEND_UT_APDU(UT_TEST_COMMAND1) to the PCD.

b) The LT waits until the PCD sends an I-block to the LT with the INF field containing the UT_TEST_COMMAND1.

c) The LT sends an erroneous S(WTX) request.

d) The PCD shall send $R(NAK)_0$.

e) The LT sends a valid S(WTX) request.

f) The PCD shall answer with S(WTX) response.

g) The LT sends I-block (containing some response UT_APDU with answer to the UT_TEST_COMMAND1) to the PCD.

h) The PCD is expected to transfer this response UT_APDU back to the UT. Check at the UT that this response UT_APDU block is correctly accepted.

Table H.47 gives part of the procedure as a scenario.

Table H.47 — Scenario H.24: Error detection and recovery of a transmission error by the PCD, Procedure 7

PCD		LT
$I(0)_0$ (INF = UT_TEST_COMMAND1)	→	
	←	S(WTX) request, wrong CRC
$R(NAK)_0$ ('BA' CID CRC or 'B2' CRC)	→	
	←	S(WTX) request
S(WTX) response	→	
	←	$I(0)_0$(INF = answer to UT_TEST_COMMAND1)

H.4.4.3.9 Procedure 8 (ISO/IEC 14443-4:2018, 7.5.7, Annex B, Scenario 17)

Use the following sequence immediately after procedure H.4.4.3.1:

a) The UT sends the SEND_UT_APDU(UT_TEST_COMMAND1) to the PCD.

b) The LT waits until the PCD sends an I-block to the LT with the INF field containing the UT_TEST_COMMAND1.

c) The LT sends an S(WTX) request.

d) The PCD shall answer with S(WTX) response.

e) The LT sends an erroneous $I(0)_0$.

f) The PCD shall send $R(NAK)_0$.

g) The LT sends a valid $I(0)_0$ with maximum timing between $R(NAK)_0$ and $I(0)_0$ to check that the PCD FWT is still extended.

h) The PCD is expected to transfer this response UT_APDU back to the UT. Check at the UT that this response UT_APDU block is correctly accepted.

Table H.48 gives part of the procedure as a scenario.

Table H.48 — Scenario H.25: Error detection and recovery of a transmission error by the PCD, Procedure 8

PCD		LT
$I(0)_0$ (INF = UT_TEST_COMMAND1)	→	
	←	S(WTX) request
S(WTX) response	→	
	←	$I(0)_0$(INF = answer to UT_TEST_COMMAND1, wrong CRC)
$R(NAK)_0$ ('BA' CID CRC or 'B2' CRC)	→	
	←	$I(0)_0$(INF = answer to UT_TEST_COMMAND1)

H.4.4.3.10 Procedure 9 (with chaining) (see ISO/IEC 14443-4:2018, Annex B, Scenario 20)

Use the following sequence immediately after procedure H.4.4.3.1:

a) The UT sends the SEND_UT_APDU(UT_TEST_COMMAND2(causing 3 chains)) to the PCD.

b) The LT waits until the PCD sends an I-block $I(1)_0$ to the LT, with the INF field the first chain of the UT_TEST_COMMAND2.

c) The LT sends an erroneous $R(ACK)_0$.

d) The PCD shall send $R(NAK)_0$.

e) The LT sends $R(ACK)_0$.

f) The PCD shall send the next block $I(1)_1$ of the chain.

g) The LT sends $R(ACK)_1$.

h) The PCD shall send the last block $I(0)_0$ of the chain .

i) The LT sends I-block (containing some response UT_APDU with answer to the UT_TEST_COMMAND2) to the PCD.

j) The PCD is expected to transfer this response UT_APDU back to the UT. Check at the UT that this response UT_APDU block is correctly accepted.

In case the UT cannot force the PCD to use 3 chains to send the TEST_COMMAND2, modify the expected procedure to reflect the test purpose which is to make sure that the block numbering and the chaining are properly executed after transmission error at step c).

Table H.49 gives part of the procedure as a scenario.

Table H.49 — Scenario H.26: Error detection and recovery of a transmission error by the PCD, Procedure 9

PCD		LT
$I(1)_0$(INF = the first chain of the UT_TEST_COMMAND2)	→	
	←	$R(ACK)_0$, wrong CRC ('AA' CID '00 00' or 'A2 00 00')[a]
$R(NAK)_0$ ('BA' CID CRC or 'B2' CRC)	→	
	←	$R(ACK)_0$ ('AA' CID CRC or 'A2' CRC)[a]
$I(1)_1$(INF = the second chain of the UT_TEST_COMMAND2)	→	
	←	$R(ACK)_1$ ('AB' CID CRC or 'A3' CRC)[a]
$I(0)_0$(INF = the last chain of the UT_TEST_COMMAND2)	→	
	←	$I(0)_0$(INF = answer to UT_TEST_COMMAND2)

[a] If the PCD frame contains a CID, the left option shall be used, else the right option shall be used.

H.4.4.3.11 Procedure 10 (with chaining) (see ISO/IEC 14443-4:2018, Annex B, Scenario 21)

Use the following sequence immediately after procedure H.4.4.3.1:

a) The UT sends SEND_UT_APDU(UT_TEST_COMMAND2(causing 3 chains)) to the PCD.

b) The LT waits until the PCD sends an I-block $I(1)_0$ to the LT, with the INF field the first chain of the UT_TEST_COMMAND2.

c) The LT sends $R(ACK)_0$.

d) The PCD shall send an I-block $I(1)_1$ to the LT.

e) The LT remains mute.

f) The PCD shall send $R(NAK)_1$.

g) The LT sends $R(ACK)_0$.

h) The PCD shall send the previous I-block $I(1)_1$ to the LT.

i) The LT sends $R(ACK)_1$.

j) The PCD shall send the last block $I(0)_0$ of the chain.

k) The LT sends an I-block (containing some response UT_APDU with answer to the UT_TEST_COMMAND2) to the PCD.

l) The PCD is expected to transfer this response UT_APDU back to the UT. Check at the UT that this response UT_APDU block is correctly accepted.

In case the UT cannot force the PCD to use 3 chains to send the TEST_COMMAND2, modify the expected procedure to reflect the test purpose which is to make sure that the block numbering and the chaining are properly executed after Mute at step e)."

Table H.50 gives part of the procedure as a scenario.

Table H.50 — Scenario H.27: Error detection and recovery of a transmission error by the PCD, Procedure 10

PCD		LT
$I(1)_0$(INF = the first chain of the UT_TEST_COMMAND2)	→	
	←	$R(ACK)_0$ ('AA' CID CRC or 'A2' CRC)[a]
$I(1)_1$(INF = the second chain of the UT_TEST_COMMAND2)	→	
	←	Mute
$R(NAK)_1$ ('BB' CID CRC or 'B3' CRC)	→	
	←	$R(ACK)_0$ ('AA' CID CRC or 'A2' CRC)[a]
$I(1)_1$(INF = the second chain of the UT_TEST_COMMAND2)	→	
	←	$R(ACK)_1$ ('AB' CID CRC or 'A3' CRC)[a]
$I(0)_0$(INF = the last chain of the UT_TEST_COMMAND2)	→	
	←	$I(0)_0$(INF = answer to UT_TEST_COMMAND2)

[a] If the PCD block contains a CID, the left option shall be used, else the right option shall be used.

H.4.4.3.12 Procedure 11 (with chaining) (see ISO/IEC 14443-4:2018, Annex B, Scenario 24)

Use the following sequence immediately after procedure H.4.4.3.1:

a) The UT sends the SEND_UT_APDU(UT_TEST_COMMAND1) to the PCD.

b) The LT waits until the PCD sends an I-block $I(0)_0$ to the LT, with the INF field containing the UT_TEST_COMMAND1.

c) The LT sends I-block $I(1)_0$ indicating chaining.

d) The PCD shall send $R(ACK)_1$.

e) The LT sends erroneous I-block $I(1)_1$ to the PCD.

f) The PCD shall send $R(ACK)_1$.

g) The LT retransmits an I-block $I(1)_1$ without error.

h) The PCD shall send $R(ACK)_0$.

i) The LT sends the last block of the chain in I-block $I(0)_0$ (of its response UT_APDU with answer to the UT_TEST_COMMAND1) to the PCD.

j) The PCD is expected to transfer the response UT_APDU, containing all chaining segments, back to the UT. Check at the UT that this response UT_APDU block is correctly accepted.

Table H.51 gives part of the procedure as a scenario.

Table H.51 — Scenario H.28: Error detection and recovery of a transmission error by the PCD, Procedure 11

PCD		LT
$I(0)_0$(INF = UT_TEST_COMMAND1)	→	
	←	$I(1)_0$(INF = the first chain of the answer to UT_TEST_COMMAND1)
$R(ACK)_1$ ('AB' CID CRC or 'A3' CRC)	→	
	←	$I(1)_1$(INF = the second chain of the answer to UT_TEST_COMMAND1, wrong CRC)
$R(ACK)_1$ ('AB' CID CRC or 'A3' CRC)	→	
	←	$I(1)_1$(INF = the second chain of the answer to UT_TEST_COMMAND1)
$R(ACK)_0$ ('AA' CID CRC or 'A2' CRC)	→	
	←	$I(0)_0$(INF = the last chain of the answer to UT_TEST_COMMAND1)

H.4.4.3.13 Procedure 12 (ISO/IEC 14443-4:2018, 7.5.5.2 rule 8)

Use the following sequence immediately after procedure H.4.4.3.1:

a) The UT sends the SEND_UT_APDU(UT_TEST_COMMAND1) to the PCD.

b) The LT waits until the PCD sends an I-block to the LT with the INF field containing the UT_TEST_COMMAND1.

c) The LT remains Mute.

d) The LT waits until the PCD sends $R(NAK)_0$ (R-block may be sent more than once).

e) The LT remains Mute.

f) The LT waits until the PCD sends a S(DESELECT) request to the LT.

g) The LT remains Mute.

h) Record the response from the PCD. The PCD shall either retransmit the S(DESELECT) request or ignore the LT.

NOTE Within this procedure, rule 8 as specified in ISO/IEC 14443-4:2018, 7.5.5.2 is tested only for S(DESELECT).

Table H.52 gives part of the procedure as a scenario.

Table H.52 — Scenario H.29: Error detection and recovery of a transmission error by the PCD, Procedure 12

PCD		LT
$I(0)_0$(INF = UT_TEST_COMMAND1)	→	
	←	Mute
$R(NAK)_0$ ('BA' CID CRC or 'B2' CRC)	→	
	←	Mute
S(DESELECT) request ('CA' CID CRC or 'C2' CRC)	→	
	←	Mute
PCD response	→	

H.4.4.4 Test report

Fill the appropriate row in Table H.60 for both Type A and Type B according to Table H.53.

Table H.53 — Result criteria for error detection and recovery

Explanation	Test result
Only when the PCD's behavior matches each procedure expected scenario	PASS
Any other case	FAIL

H.4.5 Handling of NAD during chaining

H.4.5.1 Scope

The purpose of this test is to ensure that the PCD maintains NAD in the proper way.

H.4.5.2 Apparatus

See H.1.

H.4.5.3 Procedure

Place the LT into the PCD operating volume and record the presence and the content of the PCD commands.

Use the following sequence:

a) Configure the LT as one supporting NAD.

b) Repeat procedure from Scenario H.26.

H.4.5.4 Test report

Fill the appropriate row in Table H.60 for both Type A and Type B according to Table H.54.

Table H.54 — Result criteria for handling of NAD during chaining

Explanation	Test result
Only when the PCD use the NAD only in the first packet of chaining or does not use NAD.	PASS
Any other case	FAIL

H.5 Continuous monitoring of packets sent by the PCD

H.5.1 Scope

The purpose of this test is to ensure that the PCD does not set any RFU bits in any sent frame to any value other than the default value documented for such RFU bit. Further, the test shall also ensure that no field is ever set to RFU value. The test shall also ensure that the R-block and the S-block match the protocol definitions and that rules given regarding the first byte of the packet are not violated.

H.5.2 RFU fields

RFU fields shall be continuously monitored during the testing and shall always be verified to contain the assigned default value. A test shall be FAIL and the tested PCD shall be declared non-compliant in case an RFU field is not set to its default value at any time.

H.5.3 RFU values

Functional fields shall be continuously monitored during the testing and shall always be verified to contain only functional values documented in the standard or proprietary values documented as such in the standard. A test shall be FAIL and the tested PCD shall be declared non-compliant in case a functional field is not set to said values at any time.

H.5.4 R-block

R-block shall never contain an INF field (see ISO/IEC 14443-4:2018, 7.1.2.1).

H.5.5 S-block

An S-block shall have either

1) an INF field of one byte only when it is a WTX block, or

2) an INF field of n byte(s) ($n \geq 0$) when it is a PARAMETERS block, or

3) no INF field otherwise

(see ISO/IEC 14443-4:2018, 7.1.2.1).

H.5.6 PCB

PCB byte shall contain allowed values (see ISO/IEC 14443-4:2018, 7.1.2 and ISO/IEC 14443-4:2018, Annex C).

H.5.7 Type A initialization frames

Type A initialization frames shall contain allowed values (see ISO/IEC 14443-3:2018, 6.4.1 and 6.5.3.2).

H.5.8 Apparatus

See H.1.

H.5.9 Procedure

During all test procedures and scenarios the logical content of the communication shall always be recorded.

H.5.10 Test report

Fill the appropriate rows in Table H.60 and Table H.61 for both Type A and Type B according to Table H.55.

Table H.55 — Result criteria for continuous monitoring of packets sent by the PCD

Explanation	Test result
When the PCD satisfies all of the following conditions: 1) The PCD sets default values to RFU bits in all sent frames. 2) The PCD does not set RFU value to any field. 3) The PCD does not violate the length rules of R-block and S-block. 4) The PCD does not violate the first byte coding rules of Block and Frame (as summarized in ISO/IEC 14443-4:2018, Annex C).	PASS
When the PCD satisfies at least one of the following conditions: 1) The PCD sets other than the default value to at least one RFU bit in any sent frame. 2) The PCD sets any field to a RFU value. 3) The PCD violates the length rules of R-block or S-block. 4) The PCD violates the first byte coding rules of Block and Frame (as summarized in ISO/IEC 14443-4:2018, Annex C).	FAIL

H.6 Reported results

The results of the tests shall be reported in Table H.56 to Table H.60.

Table H.56 — Type A specific timing table

No.	Parameter	Reference	Min value measured [$1/f_c$]	Test result PASS or FAIL or N/A
1	Frame Delay Time PICC to PCD	ISO/IEC 14443-3:2018, 6.2.1.2		
2	Request Guard Time	ISO/IEC 14443-3:2018, 6.2.2		
3	Start-up Frame Guard Time	ISO/IEC 14443-4:2018, 5.2.5	—	—
	SFGI = 0			
	SFGI = 1			
	SFGI = 14			

Table H.57 — Type B specific timing table

No.	Parameter	Reference	Measured values [etu] or [$1/f_s$] or [$1/f_c$] depending on the requirements		Test result PASS or FAIL or N/A
			Min	Max	
1	SOF low	ISO/IEC 14443-3:2018, 7.1.4			
2	SOF high	ISO/IEC 14443-3:2018, 7.1.4			
3	EOF low	ISO/IEC 14443-3:2018, 7.1.5			
4	Bit boundaries for	ISO/IEC 14443-3:2018, 7.1.1	—	—	—
	the falling edge(s)				
	the rising edge(s)				
5	EGT PCD to PICC	ISO/IEC 14443-3:2018, 7.1.2			

Table H.57 *(continued)*

No.	Parameter	Reference	Measured values [etu] or [1/f_s] or [1/f_c] depending on the requirements		Test result PASS or FAIL or N/A
			Min	Max	
6	TR2	ISO/IEC 14443-3:2018, 7.1.7	—	—	—
	(b3,b2) = (00)b				
	(b3,b2) = (01)b				
	(b3,b2) = (10)b				
	(b3,b2) = (11)b				
7	Start-up Frame Guard Time	ISO/IEC 14443-4:2018, 7.9.4.7	—	—	—
	SFGI = 0				
	SFGI = 1				
	SFGI = 14				

Table H.58 — Reported results for Type A specific test methods

Test method from ISO/IEC 10373-6		Scenario numbers	Test result
Clause	Parameter	ISO/IEC 10373-6	PASS or FAIL or N/A[a]
H.2.1	Frame Delay Time PICC to PCD	Scenario H.27	
H.2.2	Request Guard Time		
H.2.3	Handling of bit collision during ATQA		
H.2.4	Handling of anticollision loop	Scenario H.1	
		Scenario H.2	
		Scenario H.3	
		Scenario H.4	
H.2.5	Handling of RATS and ATS	Scenario H.5	
		Scenario H.6	
		Scenario H.30	
H.2.7	Frame size selection mechanism	Scenario H.10	
H.2.8	Handling of Start-up Frame Guard Time	Scenario H.11	
H.2.9	Handling of the CID during activation by the PCD	Scenario H.12	
H.2.10	Handling of parity bit	Scenario H.31	

[a] In case a test has several procedures indicate PASS only in case every individual procedure is PASS.

Table H.59 — Reported results for Type B specific test methods

Test method from ISO/IEC 10373-6		Scenario numbers	Test result
Clause	Parameter	ISO/IEC 10373-6	PASS or FAIL or N/A[a]
H.3.2	Frame size selection mechanism	Scenario H.13	
H.3.3	Handling of the CID during activation by the PCD	Scenario H.14	
		Scenario H.15	
H.3.4	Frame delay time PICC to PCD (TR2)	Scenario H.32	

[a] In case a test has several procedures indicate PASS only in case every individual procedure is PASS.

Table H.59 *(continued)*

Test method from ISO/IEC 10373-6		Scenario numbers	Test result
Clause	**Parameter**	**ISO/IEC 10373-6**	**PASS or FAIL or N/A[a]**
H.3.5	Handling of Start-up Frame Guard Time	Scenario H.33	
H.3.6	Type B PCD framing tests		

[a] In case a test has several procedures indicate PASS only in case every individual procedure is PASS.

Table H.60 — Reported results for test method for logical operations of the PCD

Test method from ISO/IEC 10373-6		Scenario numbers		Test result	
Clause	**Parameter**	**ISO/IEC 10373-6**	**ISO/IEC 14443-4:2018, Annex B**	**Type A PASS or FAIL or N/A[a]**	**Type B PASS or FAIL or N/A[a]**
H.4.2	Handling of the polling loop				
H.4.3	Reaction of the PCD to request for waiting time extension	Scenario H.17			
H.4.4	Error detection and recovery	Scenario H.18	Scenario 12 "Exchange of I-blocks"		
		Scenario H.19			
		Scenario H.20			
		Scenario H.21			
		Scenario H.22	Scenario 23 "PICC uses chaining"		
		Scenario H.23			
		Scenario H.24	Scenario 14 "Request for waiting time extension"		
		Scenario H.25	Scenario 17 "Request for waiting time extension"		
		Scenario H.26	Scenario 20 "PCD uses chaining"		
		Scenario H.27	Scenario 21 "PCD uses chaining"		
		Scenario H.28	Scenario 24 "PICC uses chaining"		
		Scenario H.29			
H.4.5	Handling of NAD during chaining				
H.5	Continuous monitoring of packets sent by the PCD				

[a] In case a test has several procedures, indicate PASS only when each individual procedure is PASS.

Table H.61 — PCD RFU report

Name	PCD command	RFU field/value	Value		Test result
			Default	Not allowed	PASS or FAIL or N/A
Short frame Type A	REQA/WUPA	RFU values		All values specified as RFU in ISO/IEC 14443-3:2018, Table 3	
SEL coding	SEL	RFU values		All values specified as RFU in ISO/IEC 14443-3:2018, Table 7	
AFI	REQB/WUPB	RFU values		All values specified as RFU in ISO/IEC 14443-3:2018, Table 22	
PARAM	REQB/WUPB	RFU field (b8 to b6)	(000)b	All other values	
		RFU values in number of slots (b3 to b1)		(101)b (110)b (111)b	
Param 1	ATTRIB	RFU field (b2 to b1)	(00)b	All other values	
Minimum TR0	ATTRIB	RFU values (b8 to b7)		(11)b	
Minimum TR1	ATTRIB	RFU values (b6 to b5)		(11)b	
Param 2	ATTRIB	RFU values (b4 to b1)		All values from '9'((1001)b) up to 'F'((1111)b)	
Param 3	ATTRIB	RFU field (b8 to b4)	(00000)b	All other values	
Param 4	ATTRIB	RFU field (b8 to b5)	(0000)b	All other values	
		RFU value (b4 to b1)		'F'((1111)b)	

Annex I
(normative)

High bit rate selection test methods for PCD

I.1 Scope

The purpose of this test is to verify that the PCD properly executes the procedures for selection of bit rates higher than $f_c/128$, as specified

1) for Type A in ISO/IEC 14443-4:2018, Clause 5,

2) for Type B in ISO/IEC 14443-3:2018, Clause 7,

3) for Type A or Type B using S(PARAMETERS) in ISO/IEC 14443-4:2018, Clause 9

and that the PCD properly applies the selected bit rates in both directions."

I.2 Apparatus

In this test the PCD-test-apparatus shall be configurable to change the bit rate during the test procedure. Tester shall be able to measure the bit rate used by the PCD on each stage of this test procedure.

I.3 Procedure

I.3.1 Procedure for Type A

I.3.1.1 General

Place the PCD-test-apparatus into the field of the PCD.

The following procedure shall be repeated for all values of interface byte TA(1) defined in Table I.1.

a) Run through activation sequence as defined in ISO/IEC 14443-3.

b) The PCD shall send a RATS command as defined in ISO/IEC 14443-4.

c) The PCD-test-apparatus answers with a valid ATS including TA(1) according to Table I.1.

d) The PCD may optionally send a PPS with a valid parameter setting for PPS1 byte according to Table I.1.

e) If the PCD has sent a PPS then the PCD-test-apparatus acknowledges the received PPS with a valid PPS response.

f) The PCD shall send $I(0)_0$ block using the bit rate selected. This block may also be $I(1)_0$, or R(NAK) in case of PICC presence check Method 2 a) as described in ISO/IEC 14443-4:2018, 7.5.6.2.

g) The PCD-test-apparatus sends a valid response using the bit rate selected. Check, if the answer from the PCD-test apparatus is accepted by the PCD.

NOTE Steps h) to k) may not be applicable when the PCD is embedded in a product.

h) The PCD shall send a S(DESELECT) request using the bit rate selected.

i) The PCD-test-apparatus sends a valid S(DESELECT) response using the bit rate selected. Check, if the answer from the PCD-test apparatus is accepted by the PCD.

j) The PCD shall send a valid WUPA command frame using a bit rate of $f_c/128$.

k) The PCD-test-apparatus answers with a valid ATQA.

Table I.1 — Correct behavior of PCD after ATS with TA(1)

TA(1)	Valid parameter setting for PPS1
(10000000)b	(00000000)b[a]
(10010001)b	(00000101)b, (00000000)b
(10100010)b	(00001010)b, (00000000)b
(10110011)b	(00000101)b, (00001010)b, (00000000)b
(11000100)b	(00001111)b, (00000000)b
(11010101)b	(00000101)b, (00001111)b, (00000000)b
(11100110)b	(00001010)b, (00001111)b, (00000000)b
(11110111)b	(00000101)b, (00001010)b, (00001111)b, (00000000)b
(00000000)b	(00000000)b[a]
(00000001)b	(00000001)b, (00000000)b
(00000010)b	(00000010)b, (00000000)b
(00000011)b	(00000001)b, (00000010)b, (00000000)b
(00000100)b	(00000011)b, (00000000)b
(00000101)b	(00000001)b, (00000011)b, (00000000)b
(00000110)b	(00000010)b, (00000011)b, (00000000)b
(00000111)b	(00000001)b, (00000010)b, (00000011)b, (00000000)b
(00010000)b	(00000000)b (00000100)b
(00010001)b	(00000001)b, (00000000)b (00000101)b, (00000100)b
(00010010)b	(00000010)b, (00000000)b (00000110)b, (00000100)b
(00010011)b	(00000001)b, (00000010)b, (00000000)b (00000101)b, (00000110)b, (00000100)b
(00010100)b	(00000011)b, (00000000)b (00000111)b, (00000100)b
(00010101)b	(00000001)b, (00000011)b, (00000000)b (00000101)b, (00000111)b, (00000100)b
(00010110)b	(00000010)b, (00000011)b, (00000000)b (00000110)b, (00000111)b, (00000100)b
(00010111)b	(00000001)b, (00000010)b, (00000011)b, (00000000)b (00000101)b, (00000110)b, (00000111)b, (00000100)b
(00100000)b	(00000000)b (00001000)b
(00100001)b	(00000001)b, (00000000)b (00001001)b, (00001000)b
(00100010)b	(00000010)b, (00000000)b (00001010)b, (00001000)b
(00100011)b	(00000001)b, (00000010)b, (00000000)b (00001001)b, (00001010)b, (00001000)b

[a] PPS command is useless in this case and may not be supported by the PICC.

Table I.1 *(continued)*

TA(1)	Valid parameter setting for PPS1
(00100100)b	(00000011)b, (00000000)b (00001011)b, (00001000)b
(00100101)b	(00000001)b, (00000011)b, (00000000)b (00001001)b, (00001011)b, (00001000)b
(00100110)b	(00000010)b, (00000011)b, (00000000)b (00001010)b, (00001011)b, (00001000)b
(00100111)b	(00000001)b, (00000010)b, (00000011)b, (00000000)b (00001001)b, (00001010)b, (00001011)b, (00001000)b
(00110000)b	(00000000)b (00000100)b (00001000)b
(00110001)b	(00000001)b, (00000000)b (00000101)b, (00000100)b (00001001)b, (00001000)b
(00110010)b	(00000010)b, (00000000)b (00000110)b, (00000100)b (00001010)b, (00001000)b
(00110011)b	(00000001)b, (00000010)b, (00000000)b (00000101)b, (00000110)b, (00000100)b (00001001)b, (00001010)b, (00001000)b
(00110100)b	(00000011)b, (00000000)b (00000111)b, (00000100)b (00001011)b, (00001000)b
(00110101)b	(00000001)b, (00000011)b, (00000000)b (00000101)b, (00000111)b, (00000100)b (00001001)b, (00001011)b, (00001000)b
(00110110)b	(00000010)b, (00000011)b, (00000000)b (00000110)b, (00000111)b, (00000100)b (00001010)b, (00001011)b, (00001000)b
(00110111)b	(00000001)b, (00000010)b, (00000011)b, (00000000)b (00000101)b, (00000110)b, (00000111)b, (00000100)b (00001001)b, (00001010)b, (00001011)b, (00001000)b
(01000000)b	(00000000)b (00001100)b
(01000001)b	(00000001)b, (00000000)b (00001101)b, (00001100)b
(01000010)b	(00000010)b, (00000000)b (00001110)b, (00001100)b
(01000011)b	(00000001)b, (00000010)b, (00000000)b (00001101)b, (00001110)b, (00001100)b
(01000100)b	(00000011)b, (00000000)b (00001111)b, (00001100)b
(01000101)b	(00000001)b, (00000011)b, (00000000)b (00001101)b, (00001111)b, (00001100)b
(01000110)b	(00000010)b, (00000011)b, (00000000)b (00001110)b, (00001111)b, (00001100)b
(01000111)b	(00000001)b, (00000010)b, (00000011)b, (00000000)b (00001101)b, (00001110)b, (00001111)b, (00001100)b
(01010000)b	(00000000)b (00000100)b (00001100)b

[a] PPS command is useless in this case and may not be supported by the PICC.

Table I.1 *(continued)*

TA(1)	Valid parameter setting for PPS1
(01010001)b	(00000001)b, (00000000)b (00000101)b, (00000100)b (00001101)b, (00001100)b
(01010010)b	(00000010)b, (00000000)b (00000110)b, (00000100)b (00001110)b, (00001100)b
(01010011)b	(00000001)b, (00000010)b, (00000000)b (00000101)b, (00000110)b, (00000100)b (00001101)b, (00001110)b, (00001100)b
(01010100)b	(00000011)b, (00000000)b (00000111)b, (00000100)b (00001111)b, (00001100)b
(01010101)b	(00000001)b, (00000011)b, (00000000)b (00000101)b, (00000111)b, (00000100)b (00001101)b, (00001111)b, (00001100)b
(01010110)b	(00000010)b, (00000011)b, (00000000)b (00000110)b, (00000111)b, (00000100)b (00001110)b, (00001111)b, (00001100)b
(01010111)b	(00000001)b, (00000010)b, (00000011)b, (00000000)b (00000101)b, (00000110)b, (00000111)b, (00000100)b (00001101)b, (00001110)b, (00001111)b, (00001100)b
(01100000)b	(00000000)b (00001000)b (00001100)b
(01100001)b	(00000001)b, (00000000)b (00001001)b, (00001000)b (00001101)b, (00001100)b
(01100010)b	(00000010)b, (00000000)b (00001010)b, (00001000)b (00001110)b, (00001100)b
(01100011)b	(00000001)b, (00000010)b, (00000000)b (00001001)b, (00001010)b, (00001000)b (00001101)b, (00001110)b, (00001100)b
(01100100)b	(00000011)b, (00000000)b (00001011)b, (00001000)b (00001111)b, (00001100)b
(01100101)b	(00000001)b, (00000011)b, (00000000)b (00001001)b, (00001011)b, (00001000)b (00001101)b, (00001111)b, (00001100)b
(01100110)b	(00000010)b, (00000011)b, (00000000)b (00001010)b, (00001011)b, (00001000)b (00001110)b, (00001111)b, (00001100)b
(01100111)b	(00000001)b, (00000010)b, (00000011)b, (00000000)b (00001001)b, (00001010)b, (00001011)b, (00001000)b (00001101)b, (00001110)b, (00001111)b, (00001100)b
(01110000)b	(00000000)b (00000100)b (00001000)b (00001100)b
(01110001)b	(00000001)b, (00000000)b (00000101)b, (00000100)b (00001001)b, (00001000)b (00001101)b, (00001100)b

[a] PPS command is useless in this case and may not be supported by the PICC.

Table I.1 *(continued)*

TA(1)	Valid parameter setting for PPS1
(01110010)b	(00000010)b, (00000000)b (00000110)b, (00000100)b (00001010)b, (00001000)b (00001110)b, (00001100)b
(01110011)b	(00000001)b, (00000010)b, (00000000)b (00000101)b, (00000110)b, (00000100)b (00001001)b, (00001010)b, (00001000)b (00001101)b, (00001110)b, (00001100)b
(01110100)b	(00000011)b, (00000000)b (00000111)b, (00000100)b (00001011)b, (00001000)b (00001111)b, (00001100)b
(01110101)b	(00000001)b, (00000011)b, (00000000)b (00000101)b, (00000111)b, (00000100)b (00001001)b, (00001011)b, (00001000)b (00001101)b, (00001111)b, (00001100)b
(01110110)b	(00000010)b, (00000011)b, (00000000)b (00000110)b, (00000111)b, (00000100)b (00001010)b, (00001011)b, (00001000)b (00001110)b, (00001111)b, (00001100)b
(01110111)b	(00000001)b, (00000010)b, (00000011)b, (00000000)b (00000101)b, (00000110)b, (00000111)b, (00000100)b (00001001)b, (00001010)b, (00001011)b, (00001000)b (00001101)b, (00001110)b, (00001111)b, (00001100)b

[a] PPS command is useless in this case and may not be supported by the PICC.

Table I.2 gives part of the procedure as a scenario.

Table I.2 — Scenario I.1: High bit rate selection, Type A, Procedure 1

PCD		PCD-test-apparatus
RATS command frame ('E0 01' CRC_A)	→	
	←	ATS, TA(1) according to Table I.1
Optional PPS request according to Table I.1	→	
	←	PPS response (using a bit rate of $f_c/128$)
$I(0)_0$ (using selected bit rate)	→	
	←	$I(0)_0$ (using selected bit rate)
S(DESELECT) request	→	
	←	S(DESELECT) response (using selected bit rate)
WUPA (using a bit rate of $f_c/128$)	→	
	←	ATQA (using a bit rate of $f_c/128$)

I.3.1.2 Test report

Only if the PCD behaves valid according to Scenario I.1 in each of the 72 test cases, then the test result is PASS. In any other case, the test result is FAIL.

The test report should document the bit rates chosen by the PCD in each of the 72 test cases.

I.3.2 Procedure for Type B

I.3.2.1 General

Place the PCD-test-apparatus into the field of the PCD.

The following procedure shall be repeated for all values of the protocol info byte Bit_Rate_capability defined in Table I.3:

a) The PCD shall send a valid REQB command frame.

b) The PCD-test-apparatus answers with a valid ATQB including Bit_Rate_capability byte according to Table I.3.

c) The PCD shall send an ATTRIB command with a valid parameter setting for Param 2 byte according to Table I.3.

d) The PCD-test-apparatus acknowledges the received ATTRIB with a valid Answer to ATTRIB command.

e) PCD shall send $I(0)_0$ block using the bit rate selected with Param 2. This block may also be $I(1)_0$, or R(NAK) in case of PICC presence check Method 2 a) as described in ISO/IEC 14443-4:2018, 7.5.6.2.

f) The PCD-test-apparatus sends a valid response using the bit rate selected with Param 2. Check, if the answer from the PCD-test apparatus is accepted by the PCD.

NOTE Steps g) to j) may not be applicable when the PCD is embedded in a product:

g) The PCD shall send a S(DESELECT) request using the bit rate selected.

h) The PCD-test-apparatus sends a valid S(DESELECT) response using the bit rate selected. Check, if the answer from the PCD-test apparatus is accepted by the PCD.

i) The PCD shall send a valid WUPB command frame using a bit rate of $f_c/128$.

j) PCD-test-apparatus answers with a valid ATQB including Bit_Rate_capability byte according to Table I.3.

Table I.3 — Correct behavior of PCD after ATQB

Bit_Rate_capability	Valid parameter setting for Param 2[a]
(10000000)b	(0000xxxx)b
(10010001)b	(0101xxxx)b, (0000xxxx)b
(10100010)b	(1010xxxx)b, (0000xxxx)b
(10110011)b	(0101xxxx)b, (1010xxxx)b, (0000xxxx)b
(11000100)b	(1111xxxx)b, (0000xxxx)b
(11010101)b	(0101xxxx)b, (1111xxxx)b, (0000xxxx)b
(11100110)b	(1010xxxx)b, (1111xxxx)b, (0000xxxx)b
(11110111)b	(0101xxxx)b, (1010xxxx)b, (1111xxxx)b, (0000xxxx)b
(00000000)b	(0000xxxx)b
(00000001)b	(0001xxxx)b, (0000xxxx)b
(00000010)b	(0010xxxx)b, (0000xxxx)b
(00000011)b	(0001xxxx)b, (0010xxxx)b, (0000xxxx)b
(00000100)b	(0011xxxx)b, (0000xxxx)b

[a] The least significant half byte of Param 2 is used to code the maximum frame size that can be received by the PCD.

Table I.3 *(continued)*

Bit_Rate_capability	Valid parameter setting for Param 2[a]
(00000101)b	(0001xxxx)b, (0011xxxx)b, (0000xxxx)b
(00000110)b	(0010xxxx)b, (0011xxxx)b, (0000xxxx)b
(00000111)b	(0001xxxx)b, (0010xxxx)b, (0011xxxx)b, (0000xxxx)b
(00010000)b	(0000xxxx)b (0100xxxx)b
(00010001)b	(0001xxxx)b, (0000xxxx)b (0101xxxx)b, (0100xxxx)b
(00010010)b	(0010xxxx)b, (0000xxxx)b (0110xxxx)b, (0100xxxx)b
(00010011)b	(0001xxxx)b, (0010xxxx)b, (0000xxxx)b (0101xxxx)b, (0110xxxx)b, (0100xxxx)b
(00010100)b	(0011xxxx)b, (0000xxxx)b (0111xxxx)b, (0100xxxx)b
(00010101)b	(0001xxxx)b, (0011xxxx)b, (0000xxxx)b (0101xxxx)b, (0111xxxx)b, (0100xxxx)b
(00010110)b	(0010xxxx)b, (0011xxxx)b, (0000xxxx)b (0110xxxx)b, (0111xxxx)b, (0100xxxx)b
(00010111)b	(0001xxxx)b, (0010xxxx)b, (0011xxxx)b, (0000xxxx)b (0101xxxx)b, (0110xxxx)b, (0111xxxx)b, (0100xxxx)b
(00100000)b	(0000xxxx)b (1000xxxx)b
(00100001)b	(0001xxxx)b, (0000xxxx)b (1001xxxx)b, (1000xxxx)b
(00100010)b	(0010xxxx)b, (0000xxxx)b (1010xxxx)b, (1000xxxx)b
(00100011)b	(0001xxxx)b, (0010xxxx)b, (0000xxxx)b (1001xxxx)b, (1010xxxx)b, (1000xxxx)b
(00100100)b	(0011xxxx)b, (0000xxxx)b (1011xxxx)b, (1000xxxx)b
(00100101)b	(0001xxxx)b, (0011xxxx)b, (0000xxxx)b (1001xxxx)b, (1011xxxx)b, (1000xxxx)b
(00100110)b	(0010xxxx)b, (0011xxxx)b, (0000xxxx)b (1010xxxx)b, (1011xxxx)b, (1000xxxx)b
(00100111)b	(0001xxxx)b, (0010xxxx)b, (0011xxxx)b, (0000xxxx)b (1001xxxx)b, (1010xxxx)b, (1011xxxx)b, (1000xxxx)b
(00110000)b	(0000xxxx)b (0100xxxx)b (1000xxxx)b
(00110001)b	(0001xxxx)b, (0000xxxx)b (0101xxxx)b, (0100xxxx)b (1001xxxx)b, (1000xxxx)b
(00110010)b	(0010xxxx)b, (0000xxxx)b (0110xxxx)b, (0100xxxx)b (1010xxxx)b, (1000xxxx)b
(00110011)b	(0001xxxx)b, (0010xxxx)b, (0000xxxx)b (0101xxxx)b, (0110xxxx)b, (0100xxxx)b (1001xxxx)b, (1010xxxx)b, (1000xxxx)b
(00110100)b	(0011xxxx)b, (0000xxxx)b (0111xxxx)b, (0100xxxx)b (1011xxxx)b, (1000xxxx)b

[a] The least significant half byte of Param 2 is used to code the maximum frame size that can be received by the PCD.

Table I.3 *(continued)*

Bit_Rate_capability	Valid parameter setting for Param 2[a]
(00110101)b	(0001xxxx)b, (0011xxxx)b, (0000xxxx)b (0101xxxx)b, (0111xxxx)b, (0100xxxx)b (1001xxxx)b, (1011xxxx)b, (1000xxxx)b
(00110110)b	(0010xxxx)b, (0011xxxx)b, (0000xxxx)b (0110xxxx)b, (0111xxxx)b, (0100xxxx)b (1010xxxx)b, (1011xxxx)b, (1000xxxx)b
(00110111)b	(0001xxxx)b, (0010xxxx)b, (0011xxxx)b, (0000xxxx)b (0101xxxx)b, (0110xxxx)b, (0111xxxx)b, (0100xxxx)b (1001xxxx)b, (1010xxxx)b, (1011xxxx)b, (1000xxxx)b
(01000000)b	(0000xxxx)b (1100xxxx)b
(01000001)b	(0001xxxx)b, (0000xxxx)b (1101xxxx)b, (1100xxxx)b
(01000010)b	(0010xxxx)b, (0000xxxx)b (1110xxxx)b, (1100xxxx)b
(01000011)b	(0001xxxx)b, (0010xxxx)b, (0000xxxx)b (1101xxxx)b, (1110xxxx)b, (1100xxxx)b
(01000100)b	(0011xxxx)b, (0000xxxx)b (1111xxxx)b, (1100xxxx)b
(01000101)b	(0001xxxx)b, (0011xxxx)b, (0000xxxx)b (1101xxxx)b, (1111xxxx)b, (1100xxxx)b
(01000110)b	(0010xxxx)b, (0011xxxx)b, (0000xxxx)b (1110xxxx)b, (1111xxxx)b, (1100xxxx)b
(01000111)b	(0001xxxx)b, (0010xxxx)b, (0011xxxx)b, (0000xxxx)b (1101xxxx)b, (1110xxxx)b, (1111xxxx)b, (1100xxxx)b
(01010000)b	(0000xxxx)b (0100xxxx)b (1100xxxx)b
(01010001)b	(0001xxxx)b, (0000xxxx)b (0101xxxx)b, (0100xxxx)b (1101xxxx)b, (1100xxxx)b
(01010010)b	(0010xxxx)b, (0000xxxx)b (0110xxxx)b, (0100xxxx)b (1110xxxx)b, (1100xxxx)b
(01010011)b	(0001xxxx)b, (0010xxxx)b, (0000xxxx)b (0101xxxx)b, (0110xxxx)b, (0100xxxx)b (1101xxxx)b, (1110xxxx)b, (1100xxxx)b
(01010100)b	(0011xxxx)b, (0000xxxx)b (0111xxxx)b, (0100xxxx)b (1111xxxx)b, (1100xxxx)b
(01010101)b	(0001xxxx)b, (0011xxxx)b, (0000xxxx)b (0101xxxx)b, (0111xxxx)b, (0100xxxx)b (1101xxxx)b, (1111xxxx)b, (1100xxxx)b
(01010110)b	(0010xxxx)b, (0011xxxx)b, (0000xxxx)b (0110xxxx)b, (0111xxxx)b, (0100xxxx)b (1110xxxx)b, (1111xxxx)b, (1100xxxx)b
(01010111)b	(0001xxxx)b, (0010xxxx)b, (0011xxxx)b, (0000xxxx)b (0101xxxx)b, (0110xxxx)b, (0111xxxx)b, (0100xxxx)b (1101xxxx)b, (1110xxxx)b, (1111xxxx)b, (1100xxxx)b
(01100000)b	(0000xxxx)b (1000xxxx)b (1100xxxx)b

[a] The least significant half byte of Param 2 is used to code the maximum frame size that can be received by the PCD.

Table I.3 *(continued)*

Bit_Rate_capability	Valid parameter setting for Param 2[a]
(01100001)b	(0001xxxx)b, (0000xxxx)b (1001xxxx)b, (1000xxxx)b (1101xxxx)b, (1100xxxx)b
(01100010)b	(0010xxxx)b, (0000xxxx)b (1010xxxx)b, (1000xxxx)b (1110xxxx)b, (1100xxxx)b
(01100011)b	(0001xxxx)b, (0010xxxx)b, (0000xxxx)b (1001xxxx)b, (1010xxxx)b, (1000xxxx)b (1101xxxx)b, (1110xxxx)b, (1100xxxx)b
(01100100)b	(0011xxxx)b, (0000xxxx)b (1011xxxx)b, (1000xxxx)b (1111xxxx)b, (1100xxxx)b
(01100101)b	(0001xxxx)b, (0011xxxx)b, (0000xxxx)b (1001xxxx)b, (1011xxxx)b, (1000xxxx)b (1101xxxx)b, (1111xxxx)b, (1100xxxx)b
(01100110)b	(0010xxxx)b, (0011xxxx)b, (0000xxxx)b (1010xxxx)b, (1011xxxx)b, (1000xxxx)b (1110xxxx)b, (1111xxxx)b, (1100xxxx)b
(01100111)b	(0001xxxx)b, (0010xxxx)b, (0011xxxx)b, (0000xxxx)b (1001xxxx)b, (1010xxxx)b, (1011xxxx)b, (1000xxxx)b (1101xxxx)b, (1110xxxx)b, (1111xxxx)b, (1100xxxx)b
(01110000)b	(0000xxxx)b (0100xxxx)b (1000xxxx)b (1100xxxx)b
(01110001)b	(0001xxxx)b, (0000xxxx)b (0101xxxx)b, (0100xxxx)b (1001xxxx)b, (1000xxxx)b (1101xxxx)b, (1100xxxx)b
(01110010)b	(0010xxxx)b, (0000xxxx)b (0110xxxx)b, (0100xxxx)b (1010xxxx)b, (1000xxxx)b (1110xxxx)b, (1100xxxx)b
(01110011)b	(0001xxxx)b, (0010xxxx)b, (0000xxxx)b (0101xxxx)b, (0110xxxx)b, (0100xxxx)b (1001xxxx)b, (1010xxxx)b, (1000xxxx)b (1101xxxx)b, (1110xxxx)b, (1100xxxx)b
(01110100)b	(0011xxxx)b, (0000xxxx)b (0111xxxx)b, (0100xxxx)b (1011xxxx)b, (1000xxxx)b (1111xxxx)b, (1100xxxx)b
(01110101)b	(0001xxxx)b, (0011xxxx)b, (0000xxxx)b (0101xxxx)b, (0111xxxx)b, (0100xxxx)b (1001xxxx)b, (1011xxxx)b, (1000xxxx)b (1101xxxx)b, (1111xxxx)b, (1100xxxx)b
(01110110)b	(0010xxxx)b, (0011xxxx)b, (0000xxxx)b (0110xxxx)b, (0111xxxx)b, (0100xxxx)b (1010xxxx)b, (1011xxxx)b, (1000xxxx)b (1110xxxx)b, (1111xxxx)b, (1100xxxx)b
(01110111)b	(0001xxxx)b, (0010xxxx)b, (0011xxxx)b, (0000xxxx)b (0101xxxx)b, (0110xxxx)b, (0111xxxx)b, (0100xxxx)b (1001xxxx)b, (1010xxxx)b, (1011xxxx)b, (1000xxxx)b (1101xxxx)b, (1110xxxx)b, (1111xxxx)b, (1100xxxx)b

[a] The least significant half byte of Param 2 is used to code the maximum frame size that can be received by the PCD.

Table I.4 gives part of the procedure as a scenario.

Table I.4 — Scenario I.2: High bit rate selection, Type B, Procedure 2

PCD		PCD-test-apparatus
REQB ('05 00 00' CRC_B)	→	
	←	ATQB, Bit_Rate_capability according to Table I.2
ATTRIB command, Param 2 according to Table I.2	→	
	←	Answer to ATTRIB command (using a bit rate of $f_c/128$)
$I(0)_0$ (using selected bit rate)	→	
	←	$I(0)_0$ (using selected bit rate)
S(DESELECT) request	→	
	←	S(DESELECT) response (using selected bit rate)
WUPB (using a bit rate of $f_c/128$)	→	
	←	ATQB (using a bit rate of $f_c/128$)

I.3.2.2 Test report

Only if the PCD behaves valid according to Scenario I.2 in each of the 72 test cases, then the test result is PASS. In any other case, the test result is FAIL.

The test report should document the bit rates chosen by the PCD in each of the 72 test cases.

I.3.3 Procedure for bit rate selection using S(PARAMETERS) blocks

I.3.3.1 General

Place the PCD-test-apparatus into the field of the PCD.

The following procedure shall be repeated for all values of bit rates information bytes defined in Table I.5 for PCDs using S(PARAMETERS) mechanism for bit rate selection as defined in ISO/IEC 14443-4:2018, Clause 9:

a) The UT performs the protocol activation procedure according to H.1.9.2 for Type A or H.1.9.3 for Type B.

NOTE 1 When a PCD is embedded in a product or the negotiation does not start immediately after protocol activation procedure, the method to activate the mechanism should be provided by the PCD manufacturer.

b) The PCD shall send an S(PARAMETERS) block to request bit rates parameters.

c) The PCD-test-apparatus answers with a valid S(PARAMETERS) block including bit rates information bytes according to Table I.5.

NOTE 2 Only a special set of all possible bit rates is tested. 2nd byte is always set to '00' for both directions.

d) The PCD shall send an S(PARAMETERS) block with a valid INF field containing bit rate selection bytes with exactly one bit set for bit rate from PCD to PICC and exactly one bit set for bit rate from PICC to PCD indicated by the PCD-test-apparatus in step c).

e) The PCD-test-apparatus acknowledges the received S(PARAMETERS) block with a valid S(PARAMETERS) block response.

f) PCD shall send $I(0)_0$ block using the bit rate selected. This block may also be $I(1)_0$, or R(NAK) in case of PICC presence check Method 2 a) as described in ISO/IEC 14443-4:2018, 7.5.6.2.

g) The PCD-test-apparatus sends a valid response using the bit rate selected. Check, if the answer from the PCD-test apparatus is accepted by the PCD.

NOTE 3 Steps h) to k) may not be applicable when a PCD is embedded in a product.

h) The PCD shall send an S(DESELECT) request using the bit rate selected.

i) The PCD-test-apparatus sends a valid S(DESELECT) response using the bit rate selected. Check, if the answer from the PCD-test apparatus is accepted by the PCD.

j) The PCD shall send a valid WUPA for Type A or WUPB for Type B using a bit rate of $f_c/128$.

k) The PCD-test-apparatus answers with a valid ATQA for Type A or ATQB for Type B.

Table I.5 — Supported bit rates

1st byte PCD to PICC								1st byte PICC to PCD							
b8	**b7**	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**	**b8**	**b7**	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1
0	0	0	0	0	1	1	1	0	0	0	0	0	1	1	1
0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
0	0	0	1	1	1	1	1	0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1	0	0	1	1	1	1	1	1
0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	1
0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	1
0	1	1	1	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	1	0	0	0	0	0	1	1	1
0	0	0	0	0	0	0	1	0	0	0	0	1	1	1	1
0	0	0	0	0	0	0	1	0	0	0	1	1	1	1	1
0	0	0	0	0	0	0	1	0	0	1	1	1	1	1	1
0	0	0	0	0	0	0	1	0	1	1	1	1	1	1	1
0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	1
0	0	0	0	1	0	0	1	0	0	0	0	1	0	0	1
0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
0	0	1	0	0	0	0	1	0	0	1	0	0	0	0	1
0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	1
0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1

Table I.5 *(continued)*

1st byte PCD to PICC								1st byte PICC to PCD							
b8	**b7**	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**	**b8**	**b7**	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**
0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	1
0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1
0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	1
0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1
0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1
0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1

NOTE 4 The coding of these bytes is defined in ISO/IEC 14443-4:2018, Figure 24.

Table I.6 gives part of the procedure as a scenario.

Table I.6 — Scenario I.3: Bit rate selection using S(PARAMETERS) blocks, Procedure 3

PCD		PCD-test-apparatus
S(PARAMETERS)	→	
	←	S(PARAMETERS), bit rates information according to Table I.3
S(PARAMETERS), bit rate selection with only one bit set for each direction, out of the received bit rates capabilities bytes	→	
	←	S(PARAMETERS) (using a bit rate of $f_c/128$)
$I(0)_0$ (using selected bit rate)	→	
	←	$I(0)_0$ (using selected bit rate)
S(DESELECT) request (using selected bit rate)	→	
	←	S(DESELECT) response (using selected bit rate)
WUPA for Type A, WUPB for Type B (using a bit rate of $f_c/128$)	→	
	←	ATQA for Type A, ATQB for Type B (using a bit rate of $f_c/128$)

I.3.3.2 Test report

Only if the PCD behaves valid according to Scenario I.3 in each of the 34 test cases, then the test result is PASS. In any other case, the test result is FAIL.

The test report should document the bit rates chosen by the PCD in each of the 34 test cases.

Annex J
(informative)

Program for EMD level measurements

The following program written in C language may be used to perform the EMD level measurements for PICC to PCD bit rates of $f_c/128$, $f_c/64$, $f_c/32$ and $f_c/16$.

NOTE 1 For EMD level measurements with a PICC subcarrier frequency higher than $f_c/16$, the program needs to be adapted accordingly.

NOTE 2 The output of (time, USB, LSB) can depend on compiler options and the used operating system architecture.

```
/*********************************************************************/
/*** This program calculates the upper side band (USB) and         ***/
/*** lower side band (LSB) load modulation amplitudes              ***/
/*** versus time of a PICC for the evaluation of EMD levels        ***/
/*********************************************************************/
/*** Input:                                                        ***/
/*** File in CSV format containing a table of two                  ***/
/*** columns (time and sense coils' voltage)                       ***/
/*** data format of input-file:                                    ***/
/*** - one data-point per line:                                    ***/
/***   (time[seconds], sense-coil-voltage[volts])                  ***/
/*** - contents in ASCII, no headers                               ***/
/*** - data-points shall be equidistant time                       ***/
/*** - minimum sampling rate: 100 MSamples/second                  ***/
/*** - at least 200 microsecond before start of PICC               ***/
/***   sub-carrier generation                                      ***/
/*** - at least 50 microsecond after start of PICC                 ***/
/***   sub-carrier generation                                      ***/
/*********************************************************************/
/*** Example for spreadsheet file (start in next line):            ***/
/*** (time) (voltage)                                              ***/
/*** 3.00000e-06,1.00                                              ***/
/*** 3.00200e-06,1.01                                              ***/
/***                                                              ***/
/*** Output:                                                       ***/
/*** File in CSV format containing the results                     ***/
/*** in a table of three columns (time, USB, LSB)                  ***/
/*********************************************************************/
/*** RUN:                                                          ***/
/*** "exefilename" filename[.csv]                                  ***/
/*********************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#define MAX_SAMPLES 5000000
#define MAX_LMA 100000
/*** Declare function prototype ***/
int File_info(char *, double *);
int readcsv(char *, double *, double *);
int writecsv(char*, int, double *, double *);
int Sliding_LMA(double, int, double *, double *, int, double *, double *);
void Noise_STD(int, double, double *);
/***                          ***/
/*********************************************************************/
/*** File_info function                                            ***/
/*** This function parses a file in CSV format                     ***/
/*** to determine the number of lines and sampling rate            ***/
```

```
/***                                                                    ***/
/*** Input:  Filename                                                   ***/
/*** Return: Number of samples (sample count)                           ***/
/***         0 if an error occurred                                     ***/
/************************************************************************/
int File_info(char * fname, double * pdt)
{
  int i, c;
  double t1, v1, t2, v2;
  FILE *sample_file;
  /*** Open File ***/
  if (!strchr(fname, '.')) strcat(fname, ".csv");
  if ((sample_file = fopen(fname, "r")) == NULL)
  {
    printf("Cannot open input file %s.\n", fname);
    return 0;
  }
  /*** Read two first lines to retrieve sampling rate ***/
  fscanf(sample_file, "%Lf,%Lf\n", &t1, &v1);
  if (feof(sample_file))
  {
    fclose(sample_file);
    return 0;
  }
  fscanf(sample_file, "%Lf,%Lf\n", &t2, &v2);
  if (feof(sample_file))
  {
    fclose(sample_file);
    return 0;
  }
  *pdt = t2 - t1; i = 2;
  while (!feof(sample_file))
  {
    c = fgetc(sample_file);
    if (c == '\n')
    {
      i++;
      if (i >= MAX_SAMPLES)
      {
        printf("Too many samples in input file: only %d samples \
          retained\n",i);
          break;
      }
    }
  }
  fclose(sample_file);
  return i;
}
/************************************************************************/
/*** Read CSV File Function                                             ***/
/*** This function reads the table of time and sense coil               ***/
/*** voltage from a file in CSV format                                  ***/
/***                                                                    ***/
/*** Input:  Filename                                                   ***/
/*** Return: Number of samples (sample count)                           ***/
/***         0 if an error occurred                                     ***/
/***                                                                    ***/
/*** Displays Statistics:                                               ***/
/*** Filename, sample count, sampling rate, max/min voltage             ***/
/************************************************************************/

int readcsv(char* fname, double vtime[], double vd[])
{
  double max_vd, min_vd;
  int i;
  FILE *sample_file;
  /*** Open File ***/
  if (!strchr(fname, '.')) strcat(fname, ".csv");
  if ((sample_file = fopen(fname, "r")) == NULL)
  {
    printf("Cannot open input file %s.\n", fname);
```

```
    return 0;
  }
/*** Read CSV File ***/
  max_vd = -1e-9F;
  min_vd = -max_vd;
  i = 0;
  while (!feof(sample_file))
  {
    if (i >= MAX_SAMPLES)
    {
      printf("Too many samples in input file: only %d samples read\n", i);
      break;
    }
    fscanf(sample_file, "%lf,%lf\n", &vtime[i], &vd[i]);
    if (vd[i]>max_vd) max_vd = vd[i];
    if (vd[i]<min_vd) min_vd = vd[i];
    i++;
  }
  fclose(sample_file);
  /*** Displays Statistics ***/
  printf("\n***********************************************\n");
  printf("Statistics: \n");
  printf(" Filename : %s\n", fname);
  printf(" Sample count: %d\n", i);
  printf(" Sampling rate : %1.0f MHz\n", 1e-6 / (vtime[1] - vtime[0]));
  printf(" Max(vd) : %4.0f mV\n", max_vd * 1000);
  printf(" Min(vd) : %4.0f mV\n", min_vd * 1000);
  return i;
}
/*************************************************************************/
/*** Write CSV File Function                                           ***/
/*** This function writes in a CSV format file                         ***/
/*** the result of LMA computation:                                    ***/
/*** 1st column = Time(s)                                              ***/
/*** 2nd column = Upper side band amplitude (V)                        ***/
/*** 3rd column = Lower side band amplitude (V)                        ***/
/***                                                                   ***/
/*** Return: Number of written samples                                 ***/
/***         0 if an error occurred                                    ***/
/*************************************************************************/

int writecsv(char* fname, int n_LMA, double LMA_time[], double maxSB[])
{
  int i;
  FILE *out_file;
  if ((out_file = fopen(fname, "w")) == NULL)
  {
    printf("Cannot open output file %s.\n", fname);
    return 0;
  }
  for (i = 0; i<n_LMA; i++)
  {
    fprintf(out_file, "%7.4E,%7.4E\n", LMA_time[i], maxSB[i]);
  }
  fclose(out_file);
  return i;
}
/*************************************************************************/
/*** Sliding_LMA : Load Modulation Amplitude versus Time               ***/
/*** This function calculates Upper side band and                      ***/
/*** Lower side band amplitudes as a function of time                  ***/
/***                                                                   ***/
/*** Arguments:                                                        ***/
/*** fc = carrier frequency (Hz)                                       ***/
/*** count = number of input signal samples                            ***/
/*** vtime[] = input signal time array                                 ***/
/*** vd[] = input signal voltage array                                 ***/
/*** lout = max. size of following arrays                              ***/
/*** LMA_time[] = Times to which LMAs are computed                     ***/
/*** USB[] = load modulation amplitude at fc+fs                        ***/
/*** LSB[] = load modulation amplitude at fc-fs                        ***/
```

```
/***                                                                    ***/
/*** Return value: Number of computed LMA                              ***/
/**************************************************************************/

int Sliding_LMA(double fc, int count, double vtime[], double vd[], int
  lout, double LMA_time[], double maxSB[])
{
  double USB, LSB;
  double c1_real, c1_imag;
  double c2_real, c2_imag;
  double w0, wu, wl, dt;
  double Wb; /* Bartlett window coefficient */
  int i, j, k = 0;
  int N_data; /* Time window size*/
  int N_over; /* Overlap */
  int N_middle; /* Half window size */
  double *Yuc, *Yus, *Ylc, *Yls; /* Phase factors */
  double pi; /* pi=3.14.... */
  double sum_Wb = 0; /* Sum of Bartlett coeff. */
  double cf; /* correction factor of the Bartlett window */
  pi = (double)atan(1.0) * 4; /* calculate pi */
  w0 = (double)(fc*2.0)*pi; /* carrier 13.56 MHz */
  wu = (double)(1.0 + 1.0 / 16.0)*w0; /* upper sideband 14.41 MHz */
  wl = (double)(1.0 - 1.0 / 16.0)*w0; /* lower sideband 12.71 MHz */
  /*** Time window ***/
  /*** Note: (vtime[2]-vtime[1]) is the scope sampling rate ***/
  dt = vtime[2] - vtime[1];
  /*** Number of samples for two subcarrier periods ***/
  N_data = (int)(0.5 + 2 * 16.0F / dt / fc);
  N_middle = (int)(0.5 + N_data / 2);
  N_over = (int)(0.5 + 1.0 / dt / fc); /* Overlap of 1/fc */
  /*** Allocate memory ***/
  Yuc = (double *)malloc(N_data * sizeof(double));
  if (Yuc == NULL)
  {
    printf("Cannot allocate memory");
    return 0;
  }
  Yus = (double *)malloc(N_data * sizeof(double));
  if (Yus == NULL)
  {
    printf("Cannot allocate memory");
    free(Yuc);
    return 0;
  }
  Ylc = (double *)malloc(N_data * sizeof(double));
  if (Ylc == NULL)
  {
    printf("Cannot allocate memory");
    free(Yuc); free(Yus);
    return 0;
  }
  Yls = (double *)malloc(N_data * sizeof(double));
  if (Yls == NULL)
  {
    printf("Cannot allocate memory");
    free(Yuc); free(Yus); free(Ylc);
    return 0;
  }
  /*** Calculate apodization window and phase factors ***/
  for (i = 0; i<N_data; i++)
  {
    /*** Bartlett window ***/
    if ((N_data & 1) == 0)
    {
    /*** N_data is even ***/
    if (i < (int)(N_data / 2))
    {
    Wb = 2.0F*i / (double)(N_data - 1);
    }
  else
```

```
    {
      Wb = 2.0F*(N_data - i - 1) / (double)(N_data - 1);
    }
  }
  else
  {
    /*N_data is odd */if (i <= (int)(0.001 + (N_data - 1) / 2))
    {
      Wb = 2.0F*i / (double)(N_data - 1);
    }
    else
    {
      Wb = 2.0F - 2.0F*i / (double)(N_data - 1);
    }
  }
  Yuc[i] = (double)cos(wu*i*dt)*Wb;
  Yus[i] = (double)sin(wu*i*dt)*Wb;
  Ylc[i] = (double)cos(wl*i*dt)*Wb;
  Yls[i] = (double)sin(wl*i*dt)*Wb;
  sum_Wb += Wb;
  }
  cf = N_data / sum_Wb;
  /*** DFT ***/
  for (j = 0; j<count - N_data; j = j + N_over)
  {
    c1_real = 0; /* real part of the up. sideband fourier coefficient */
    c1_imag = 0; /* imag part of the up. sideband fourier coefficient */
    c2_real = 0; /* real part of the lo. sideband fourier coefficient */
    c2_imag = 0; /* imag part of the lo. sideband fourier coefficient */
    for (i = 0; i<N_data; i++)
    {
      c1_real = c1_real + vd[i + j] * Yuc[i];
      c1_imag = c1_imag + vd[i + j] * Yus[i];
      c2_real = c2_real + vd[i + j] * Ylc[i];
      c2_imag = c2_imag + vd[i + j] * Yls[i];
    }
    /*** DFT scale ***/
    c1_real = 2.0F*cf*c1_real / (double)N_data;
    c1_imag = 2.0F*cf*c1_imag / (double)N_data;
    c2_real = 2.0F*cf*c2_real / (double)N_data;
    c2_imag = 2.0F*cf*c2_imag / (double)N_data;
    /*** Absolute fourier coefficient ***/
    USB = (double)sqrt(c1_real*c1_real + c1_imag * c1_imag);
    LSB = (double)sqrt(c2_real*c2_real + c2_imag * c2_imag);
    maxSB[k] = (double)(USB > LSB) ? USB : LSB;
    /*** Half window time ***/
    LMA_time[k] = vtime[j + N_middle]; /* Half window time */
    k++;
    if (k > lout) break; /* stop if array size is reached*/
  }
  free(Yuc);
  free(Yus); free(Ylc);
  free(Yls);
  return k;
}
/*********************************************************************/
/*** Noise_STD : Noise standard deviation                          ***/
/*** This function calculates the standard deviations              ***/
/*** at fc+fs and fc-fs as required by the noise precondition test ***/
/*** Results are meaningful only when the sense coil's             ***/
/*** signal is recorded with a reference PICC.                     ***/
/***                                                              ***/
/*** Arguments:                                                    ***/
/*** n_LMA = number of input values                                ***/
/*** USB[] = load modulation amplitude at fc+fs                    ***/
/*** LSB[] = load modulation amplitude at fc-fs                    ***/
/*** pSTD_USB= standard deviation at fc+fs                         ***/
/*** pSTD_LSB= standard deviation at fc-fs                         ***/
/*********************************************************************/

void Noise_STD(int n_LMA, double maxSB[], double *pSTD_maxSB)
```

```
{
  double P_maxSB = 0;
  int i;
  /*** Square summation ***/
  for (i = 0; i<n_LMA; i++)
  {
    P_maxSB += maxSB[i] * maxSB[i];
  }
  *pSTD_maxSB = sqrt(P_maxSB / n_LMA);
}
/*********************************************************************/
/*** MAIN program                                                  ***/
/*********************************************************************/

int main(int argc, char *argv[])
{
  char fname[256];
  char fout[256];
  int sample_count;
  int lout;  /*Maximum length of result arrays */
  int n_LMA; /* Number of computed LMA         */
  int status;
  double fc; /* Carrier frequency              */
  double std_USB, std_LSB, std_maxSB, dt;
  double *pTime, *pVolts, *pLMA_time, *pUSB, *pLSB, *pMaxSB;
  fc = 13.56e6;
  printf("\n");
  printf("************************************************************\n");
  printf("**** Program for EMD level measurements ******************\n");
  printf("**** Version: January 2019 *******************************\n");
  printf("**** According to ISO/IEC 10373-6 ************************\n");
  printf("************************************************************\n");
          if (argc > 1)
          {
            /*** First input parameter is taken as input file name ***/
            strcpy(fname, argv[1]);
          }
          else
          {
            /*** No input parameter ***/
            printf("\nCSV File name :");
            scanf("%s", fname);
          }
  if (!strchr(fname, '.')) strcat(fname, ".csv");
  if (!(sample_count = File_info(fname, &dt))) return 0; lout = (int)(sample_count / (int)
(0.5 + 1.0 / dt / fc));
  if (lout > MAX_LMA) lout = MAX_LMA;
  /*** Start of memory allocation ***/
  pTime = (double *)malloc(sample_count * sizeof(double));
  if (pTime == NULL)
  {
    printf("Cannot allocate memory");
    return 0;
  }
  pVolts = (double *)malloc(sample_count * sizeof(double));
  if (pVolts == NULL)
  {
    printf("Cannot allocate memory");
    free(pTime);
    return 0;
  }
  pMaxSB = (double *)malloc(lout * sizeof(double));
  if (pMaxSB == NULL)
  {
    printf("Cannot allocate memory");
    free(pTime); free(pVolts); free(pMaxSB);
    return 0;
  }
  pLMA_time = (double *)malloc(lout * sizeof(double));
  if (pLMA_time == NULL)
  {
```

```
    printf("Cannot allocate memory");
    free(pTime); free(pVolts); free(pMaxSB);
    return 0;
  } /* End of memory allocation */
  /*** Reading data ***/
  if (!(sample_count = readcsv(fname, pTime, pVolts))) return 0;
  if
    (!(n_LMA = Sliding_LMA(fc, sample_count, pTime, pVolts, lout, pLMA_time, pMaxSB)
      )) return 0;
  /*** Processing data ***/
  strcpy(fout, "LMA_");
  strcat(fout, fname);
  /*** writing results in a file ***/
  status = writecsv(fout, n_LMA, pLMA_time, pMaxSB);
  /*** evaluating noise floor ***/
  Noise_STD(n_LMA, pMaxSB, &std_maxSB);
  /*** Result display ***/
  printf("\n");
  printf("*********************************************\n");
  printf(" Noise floor : \n");
  printf(" standard deviation at max Side Band: %7.3f mV\n", std_maxSB * 1000);
  printf(" Note: Displayed results are meaningful only when\n");
  printf(" the sense coil's signal is recorded with a\n");
  printf(" Reference PICC.\n");
  printf("*********************************************\n");
  free(pTime);
  free(pVolts);
  free(pLMA_time);
  free(pMaxSB);
  return 1;
}
```

Annex K
(normative)

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

K.1 Overview

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

NOTE Future revisions of ISO/IEC 14443 (all parts) and this document may specify new NPV tolerance and phase noise values with corresponding test methods.

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

K.2.1 PCD Tests

K.2.1.1 General

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

K.2.1.2 PCD phase range and waveform characteristics

K.2.1.2.1 Purpose

This test is used to determine the PR 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.

K.2.1.2.2 Procedure

Apply the procedure defined in 7.1.4.2 with the following adaptions:

1) After the activation of a bit rate of $3f_c/4$, f_c, $3f_c/2$ or $2f_c$ from PCD to PICC the PCD shall transmit an $I(0)_0$(TEST_COMMAND1(1)).

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

K.2.1.2.3 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.

NOTE K.3.13 gives some example test reports.

K.2.2 PICC Tests

K.2.2.1 PICC reception

K.2.2.1.1 Purpose

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

K.2.2.1.2 Conditions

Four test conditions are defined at the border of the PICC signal parameters as defined in ISO/IEC 14443-2. A low pass filtered pseudo random white noise as defined in K.4.3 is added to the transmitted APVs such that the normalized differential phase noise (rms) is the maximum value as defined in ISO/IEC 14443-2. The test conditions are created using the Test PCD assembly in combination with digital pre-conditioning of the transmitted APVs as shown in K.4:

1) condition 1: 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;

2) condition 2: 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;

3) condition 3: 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;

4) condition 4: 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.

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

NOTE 2 K.4 informatively describes how to create the above four conditions in the base-band domain (on the complex envelope of the signal).

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

K.2.2.1.3 Procedure

For each optional PCD to PICC bit rate supported by the PICC, the PICC shall operate under each of the test conditions defined in K.2.2.1.2 after selection of that optional PCD to PICC bit rate. This PICC shall respond correctly to an I-block transmitted at that optional PCD to PICC bit rate.

For a frame size higher than 256 bytes, a frame with error correction as defined in ISO/IEC 14443-4 should be used.

K.2.2.1.4 Test report

The test report shall confirm the intended operation at the bit rates under test.

The used test conditions shall be mentioned in the test report.

K.3 PCD waveform characteristics analysis tool for bit rates of $3f_c/4$, f_c, $3f_c/2$ and $2f_c$

K.3.1 Overview

The working principle of the analysis tool for bit rates of $3f_c/4$, f_c, $3f_c/2$ and $2f_c$ is illustrated in Figure K.1.

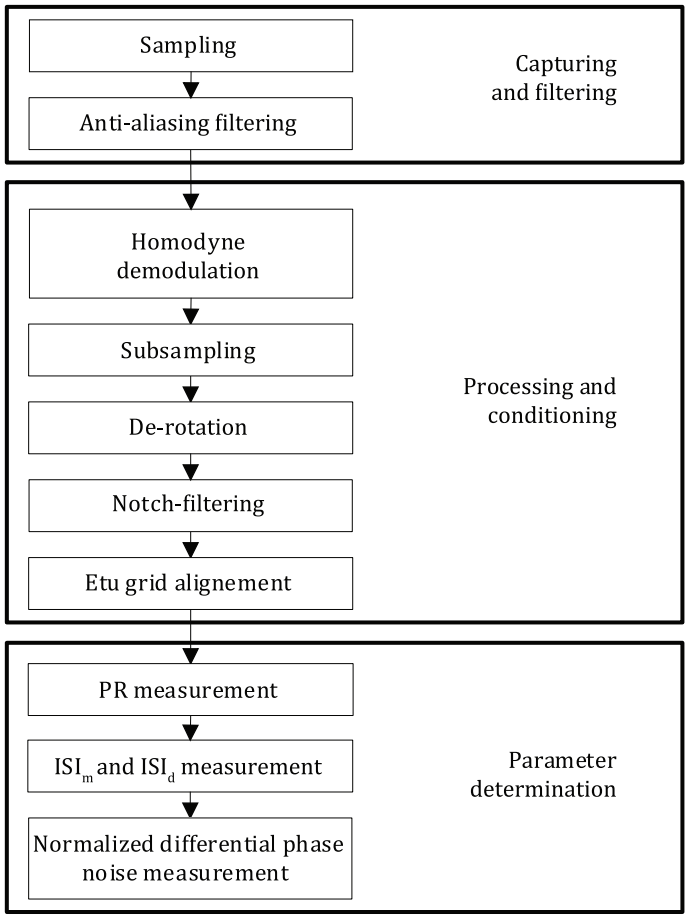


Figure K.1 — Block diagram of the analysis tool for bit rates of $3f_c/4$, f_c, $3f_c/2$ and $2f_c$

Each block is separately described in K.3.2 to K.3.11.

K.3.2 Sampling

The oscilloscope used for signal capturing shall fulfill the requirements defined in 5.2. The time and voltage data of at least 1 000 non-modulated carrier periods followed by one data frame, followed by at least 10 non-modulated carrier periods (see illustration in Figure K.2) shall be transferred to a suitable computer.

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

K.3.3 Anti-aliasing filtering

A 4th order, Butterworth type low pass filter with 3 dB cut off frequency at 120 MHz shall be used for filtering higher frequency components The filter characteristic is illustrated in Figure K.3.

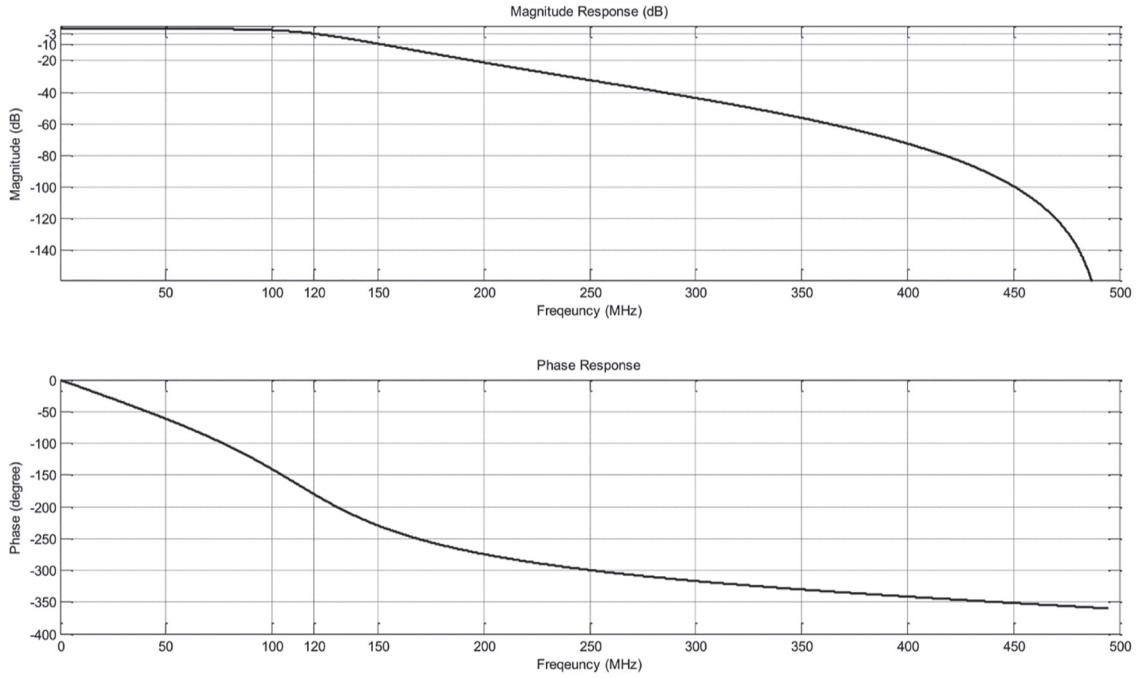


Figure K.3 — Anti-aliasing filter characteristics

K.3.4 Homodyne 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 K.4).

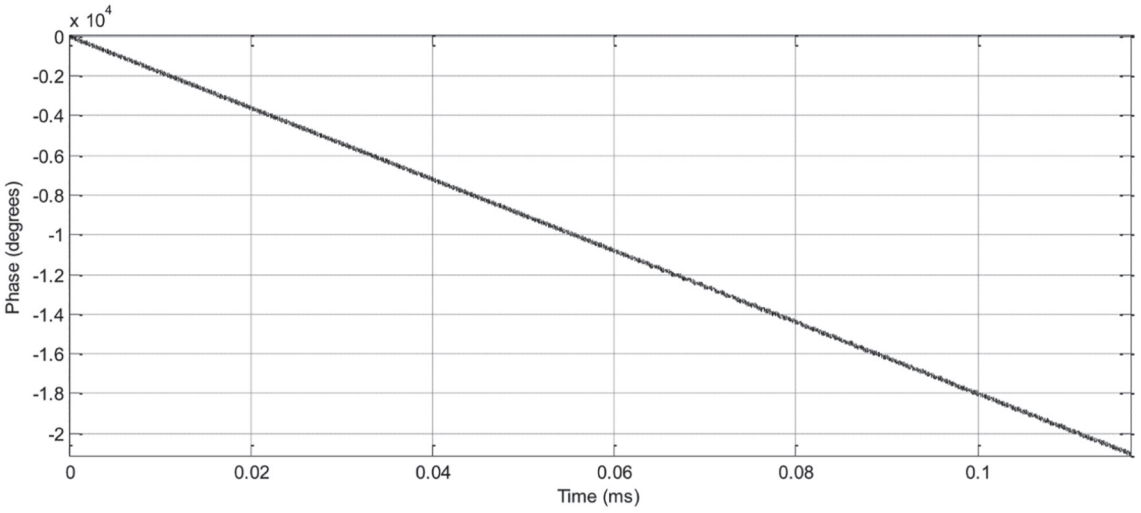


Figure K.4 — Example phase signal over time after homodyne demodulation

K.3.5 Subsampling

The phase signal shall be sub-sampled to an integer number multiple of f_c using linear interpolation. The integer number shall be at least 32.

K.3.6 De-rotation

This phase signal over time is continuously changing due to the difference between the modulated RF carrier frequency and the demodulator frequency. This frequency mismatch is computed from the constant phase slope of the phase signal during the time of the non-modulated carrier. The complete phase signal is multiplied by a carrier signal whose frequency is the computed frequency difference (see Figure K.5).

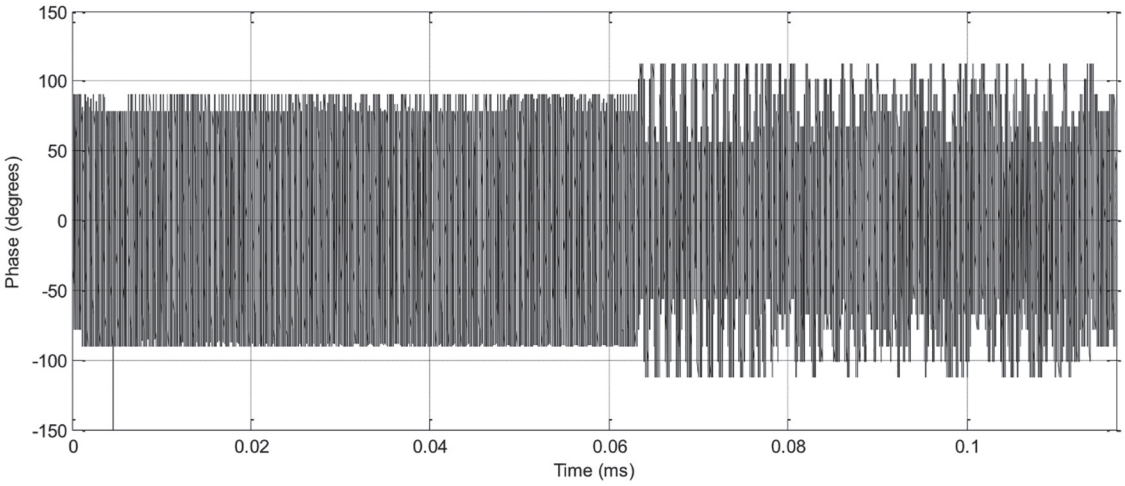


Figure K.5 — Example phase signal after de-rotation

K.3.7 Notch-filtering

The phase signal contains the second harmonic due to demodulation. The phase signal shall be smoothed with a moving average filter having a filter period of $2/f_c$ (see Figure K.6).

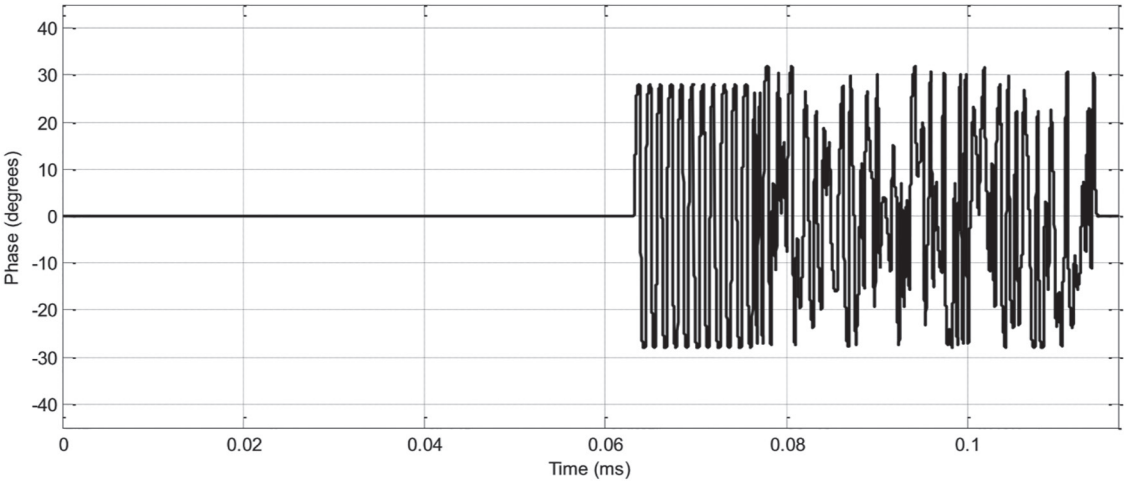


Figure K.6 — Example phase signal after filtering

K.3.8 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 using the method defined in ISO/IEC 14443-2. The etu grid alignment is carried out by maximizing the computed correlation, using the phase signal and the reference phase signal.

K.3.9 Phase range measurement

The PR parameter shall be determined as defined in ISO/IEC 14443-2.

K.3.10 Intersymbol interference measurement

The ISIm and ISId 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. ISIm and ISId values shall be computed for every sampling time within the last carrier period of an etu. The maximum ISIm value shall be selected with the related ISId.

K.3.11 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.

K.3.12 Program of the PCD waveform characteristics analysis tool for bit rates of $3f_c/4$, f_c, $3f_c/2$ and $2f_c$

The following program written in ANSI C language gives an example for the implementation of the analysis tool for bit rates of $3f_c/4$, f_c, $3f_c/2$ and $2f_c$.

This C program consists of 7 files which should be placed in the same folder.

```
/*****************************************************************/
/*** psk_defines.h                                             ***/
/***  DESCRIPTION:                                             ***/
/***    Constants and LUTs for VHBR PSK wave shape tool        ***/
/*****************************************************************/

#ifndef PSK_DEFINES_H
#define PSK_DEFINES_H

#include "psk_types.h"

#define MAX_SAMPLES 50000

#define PSK_ERR_OK                       0 /**< Successful termination */
#define PSK_ERR_READ_FILE               -1 /**< File not found or no read permission */
#define PSK_ERR_PARAMETER               -2 /**< Parameter of function is invalid or
unexpected */
#define PSK_ERR_OUT_OF_MEM              -3 /**< Memory allocation failed */
#define PSK_ERR_INVALID_SAMPLE_RATE     -4 /**< Sample rate of signal is not supported */
#define PSK_ERR_INVALID_SAMPLE_VEC      -5 /**< Sample vector could not be downsampled */
#define PSK_ERR_SIGNAL_TOO_SHORT        -6 /**< Insufficient amount of input data for
calculation */
#define PSK_SYMBOL_GRID_ALIGNMENT_FAIL  -7 /**< Grid alignment not found during analysis
*/
#define PSK_ERR_SIGNAL_LEN_MISMATCH     -8 /**< Unsupported signal length */

#ifndef M_PI
#define M_PI 3.14159265358979323846264338327795
#endif

#ifndef NULL
#define NULL 0
#endif

// carrier frequency [Hz]
#define FC 13560000
// internal sample frequency of 32 times FC [Hz]
#define FS_INT 433920000
// index of last unmodulated input sample is 480 * FS_INT / FC
#define IDX_UNMOD 15360
```

```
#define MIN_NUM_SAMPLES  30000
#define MAX_NUM_SAMPLES 900000

static const psk_uint32 PSK8[] = {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, 1, 7, 1, 7, 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};

static const psk_uint32 PSK16[] = {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, 1, 15, 1, 15, 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};

static const psk_int32  SOC_PSK8_deg[] = {  0,  24,  24, -24, -24,  24,  24,
                                          -24, -24,  24,  24, -24, -24,  24,
                                           24, -24, -24,  24,  24, -24, -24,
                                           24,  24, -24, -24,  24,  24, -24,
                                          -24,  24,  24, -24, -24,  24,  24,
                                          -24, -24,  24,  24, -24, -24,  24,
                                           24, -24, -24,  24, -24,  24, -24,
                                           32,  32, -24,   8, -16,  24,  -8,
                                            8, -16,  16,  16,  16, -24,  24,
                                           32,   8,  -8,  16,   8,  -8,  16,
                                            8, -16,  32, -24,  16,   8,   8,
                                          -24, -16,   0,  -8, -16,  24, -16,
                                           -8,  16, -16,  24,   8,   0,  32,
                                           16,  32, -16, -16, -24,  32,  -8,
                                          -24,   8, -24,   8,  32,   8, -16,
                                          -16,  24,  24,  32, -16,   0,  32,
                                          -16,   8,  -8, -16,  24,  24,  24,
                                           16, -16, -24,  32, -24,  32, -24,
                                            8,  24,  16,   0,  16,  24,  -8,
                                          -24,   0,  32,   8,   8,  16,   8, 0};

static const psk_int32  SOC_PSK16_deg[] = {  0,  28,  28, -28, -28,  28,  28,
                                           -28, -28,  28,  28, -28, -28,  28,
                                            28, -28, -28,  28,  28, -28, -28,
                                            28,  28, -28, -28,  28,  28, -28,
                                           -28,  28,  28, -28, -28,  28,  28,
                                           -28, -28,  28,  28, -28, -28,  28,
                                            28, -28, -28,  28, -28,  28, -28,
                                            32,  32, -28,   8, -12,  32,   0,
                                            16,  -8,  28,  32, -28,  -4, -20,
                                           -12,  28,  16, -20, -24,  24, -12,
                                           -20,  20,   4,  16,  -8, -16, -16,
                                            16,  28, -20, -28,  32,   8, -28,
                                           -16,  12, -16,  28,  16,   8, -20,
                                            32, -12,   4,   4,  -4, -12,  16,
                                             4, -28,   8, -20,   4, -16,  28,
                                            32,   8,  12,  20, -24,  -4,  32,
                                           -16,   8,  -8, -12,  32, -28, -24,
                                           -28,   8, -28,  32, -28,  32, -28,
                                             4,  24,  16,   0,  20,  32,   4,
                                          -12,  12, -20,  24,  28, -24, -28, 32};
```

```
static const psk_uint32 PSK_len = 141;

// output data types
enum TYPE
{
  COMPLEX,
  DOUBLE,
  INTEGER,
  BUTTERCFS
};

/*****************************************************************************/
/**
 *  The order of the input psk signal is either 16 or 8
 */
extern psk_uint32 ORDER;

/*****************************************************************************/
/**
 *  The bit rate of the input signal is a user defined parameter and can be
 *  3/4 * fc
 *        fc
 *  3/2 * fc
 *  2   * fc
 *  where fc is the carrier frequency as defined in #FC.
 */
extern psk_double BIT_RATE;

/*****************************************************************************/
/**
 *  The phase range is the difference between the highest an lowest phase value
 *  and can be either 56 deg or 60 deg
 */
extern psk_uint32 PR;

/*****************************************************************************/
/**
 *  The elementary phase interval. 8 deg or 4 deg depending on #ORDER
 */
extern psk_uint32 EPI;

/*****************************************************************************/
/**
 *  The elementary time unit is always a multiple of #FC.
 *  2/FC for bit rates 1.5*FC and 2.0*FC and 4/FC else
 */
extern psk_double ETU;

#endif // PSK_DEFINES_H

/*****************************************************************************/
/***  psk_types.h                                                          ***/
/***  DESCRIPTION:                                                         ***/
/***    Definition of used types in psk analysis tool                      ***/
/***                                                                       ***/
/*****************************************************************************/

#ifndef PSK_TYPES_H
#define PSK_TYPES_H

#define RE(z) ( (z).re )
#define IM(z) ( (z).im )
#define ABS(a) ( (a > 0) ? (a) : (-a) )
#define MAX(a, b) ( ( (a) > (b) ) ? (a) : (b) )

#define BUTTER_SIZE_A 5
#define BUTTER_SIZE_B 5

typedef double psk_double;
typedef int psk_int32;
typedef unsigned int psk_uint32;
```

```
typedef struct
{
  psk_double re;
  psk_double im;
} psk_complex;

typedef struct
{
  psk_double a[BUTTER_SIZE_A];
  psk_double b[BUTTER_SIZE_B];
} psk_butter_coefs;

#endif // PSK_TYPES_H

/*****************************************************************************/
/*** psk_math.h                                                            ***/
/***  DESCIRPTION: header of psk_math.c                                    ***/
/***  It contains the function declaration of used mathematical            ***/
/***  functions  for the PSK waveform characteristics analysis tool        ***/
/*****************************************************************************/

#ifndef PSK_MATH_H
#define PSK_MATH_H

#include "psk_types.h"

/*****************************************************************************/
/**
 * psk_mean
 *  calculate the arithmetic mean of a given vector
 *  @param vec Calculate the mean of the values in this vector
 *  @param len The vector's number of elements
 *  @return The aritmethic mean or zero, if len < 2
 */
psk_double psk_mean( psk_double* vec /*[in]*/, psk_uint32 len /*[in]*/);

/*****************************************************************************/
/**
 * psk_cmpl_mean
 *  calculate the arithmetic mean of a given vector for both, real and
 *  imaginary parts. The result is also a complex number
 *  @param vec Calculate the mean of the values in this vector
 *  @param len The vector's number of elements
 *  @return The aritmethic mean or zero, if len < 2
 */
psk_complex psk_cmpl_mean( psk_complex* vec/*[in]*/, psk_uint32 len /*[in]*/);

/*****************************************************************************/
/**
 * psk_diff
 *  The resulting vector's elements are the differences of two consecutive
 *  elements of a given vector. The resulting vector has a length of len-1
 *  @param vec Calculate the consecutive differences of this vector's values
 *  @param len The vector's number of elements
 *  @return The aritmethic mean or zero, if len < 2
 */
psk_double* psk_diff( psk_double* vec /*[in]*/, psk_uint32 len /*[in]*/);

/*****************************************************************************/
/**
 * psk_max
 *  Find the maximal value in a vector and it's index
 *  @param vec An array of values
 *  @param vec_len The vector's number of elements
 *  @param max_val Pointer where to store the maximum's value
 *  @param max_idx Pointer where to store the maximum's index
 */
void psk_max( psk_double* vec,       /*[in]*/
              psk_uint32 vec_len,    /*[in]*/
              psk_double* max_val,   /*[out]*/
```

```
              psk_uint32* max_idx ); /*[out]*/

/*****************************************************************************/
/**
 * psk_min
 *  Find the minimal value in a vector and it's index
 *  @param vec An array of values
 *  @param vec_len The vector's number of elements
 *  @param min_val Pointer where to store the minimum's value
 *  @param min_idx Pointer where to store the minimum's index
 */
void psk_min( psk_double* vec,       /*[in]*/
              psk_uint32 vec_len,    /*[in]*/
              psk_double* min_val,   /*[out]*/
              psk_uint32* min_idx ); /*[out]*/

/*****************************************************************************/
/**
 * psk_add
 *  Calculate the sum of two complex numbers
 *  @param a First summand
 *  @param b Second summand
 *  @return The complex result
 */
psk_complex psk_add( psk_complex a /*[in]*/, psk_complex b /*[in]*/ );

/*****************************************************************************/
/**
 * psk_sub
 *  Calculate the difference of two complex numbers
 *  @param a Minuend
 *  @param b Subtrahend
 *  @return The complex result
 */
psk_complex psk_sub( psk_complex a /*[in]*/, psk_complex b /*[in]*/ );

/*****************************************************************************/
/**
 * psk_cmpl_mult
 *  Calculate the product of two complex numbers
 *  @param a First factor
 *  @param b Second factor
 *  @return The complex result
 */
psk_complex psk_cmpl_mult( psk_complex a /*[in]*/, psk_complex b /*[in]*/ );

/*****************************************************************************/
/**
 * psk_cmpl_div
 *  Calculate the quotient of two complex numbers
 *  @param a Dividend
 *  @param b Divisor
 *  @return The complex result
 */
psk_complex psk_cmpl_div( psk_complex a /*[in]*/, psk_complex b /*[in]*/ );

/*****************************************************************************/
/**
 * psk_cmpl_conj
 *  Get the complex conjugate of a number
 *  @param a The complex number
 *  @return The complex conjugate of a
 */
psk_complex psk_cmpl_conj( psk_complex a /*[in]*/ );

/*****************************************************************************/
/**
 * psk_abs
 *  Get the absolute value of a complex number: sqrt(RE^2+IM^2)
 *  @param num The complex number
 *  @return The absolute value
```

```
 */
psk_double psk_abs( psk_complex num /*[in]*/ );

/*****************************************************************/
/**
 * psk_cmpl_vec_mult
 *  Calculate the product of vector elements with the same index and return
 *  the result in a vector. This function makes use of #psk_cmpl_mult
 *  The two vectors must have the same length
 *  @param a Factors of first vector
 *  @param b Factros of second vector
 *  @param len The number of elments. Must be the same for a and b
 *  @return The complex product of a and b
 */
psk_complex* psk_cmpl_vec_mult( psk_complex* a,    /*[in]*/
                                psk_complex* b,    /*[in]*/
                                psk_uint32 len );  /*[in]*/

/*****************************************************************/
/**
 * psk_vec_abs
 *  Apply #psk_abs on every complex element of a vector
 *  @param a The complex input values
 *  @param len The number of elements in vector a
 *  @return The absolute values
 */
psk_double* psk_vec_abs( psk_complex* a /*[in]*/, psk_uint32 len /*[in]*/ );

/*****************************************************************/
/**
 * psk_cmpl_vec_conj
 *  Apply #psk_cmpl_conj on every complex element of a vector
 *  @param vec The complex input values
 *  @param len The number of elements in vector vec
 *  @return The complex conjugate of vec
 */
psk_complex* psk_cmpl_vec_conj( psk_complex* vec, /*[in]*/
                                psk_uint32 len);  /*[in]*/

/*****************************************************************/
/**
 * psk_vec_angle
 *  Apply the built in function atan2 on every complex element of a vector
 *  @param vec The complex input values
 *  @param len The number of elements in vector vec
 *  @return The angle of every input element
 */
psk_double* psk_vec_angle( psk_complex* vec /*[in]*/, psk_uint32 len /*[in]*/ );

/*****************************************************************/
/**
 * psk_variance
 *  Calculate the variance of the elements in a vector
 *  @param vec The real number input values
 *  @param len The number of elements in vector vec
 *  @return The variance of the vector's elements
 */
psk_double psk_variance( psk_double* vec /*[in]*/, psk_uint32 len /*[in]*/ );

/*****************************************************************/
/**
 * psk_std
 *  Calculate the standard deviation of the elements in a vector
 *  This function uses #psk_variance and returns it's square root
 *  @param vec The real number input values
 *  @param len The number of elements in vector vec
 *  @return The standard deviation of the vector's elements
 */
psk_double psk_std( psk_double* vec /*[in]*/, psk_uint32 len /*[in]*/ );

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

```
/**
 * psk_normalize
 *  Normalize a vector's elements. This function uses #psk_vec_abs and #psk_max
 *  @param vec The real number input values.
 *  @param len The number of elements in vector vec
 *  @return The normalized vector
 */
psk_double* psk_normalize( psk_double* vec,  /*[in]*/
                           psk_uint32 len ); /*[in]*/

/*********************************************************************/
/**
 * psk_cmpl_normalize
 *  Normalize a vector's complex elements both real and imaginary part separatly
 *  @param vec The complex number input values.
 *  @param len The number of elements in vector vec
 *  @return The normalized vector
 */
psk_complex* psk_cmpl_normalize( psk_complex* vec, /*[in]*/
                                 psk_uint32 len ); /*[in]*/

/*********************************************************************/
/**
 * psk_linspace
 *  Build a vector with a given start and stop value and defined step size
 *  @param start The value of the first element
 *  @param step The step size. Difference between two consecutive elements
 *  @param stop The value of the last element
 *  @param len Pointer where to save the length of the vector
 *  @return The resulting vector or NULL, if start >= stop
 */
psk_double* psk_linspace( psk_double start,  /*[in]*/
                          psk_double step,   /*[in]*/
                          psk_double stop,   /*[in]*/
                          psk_uint32* len ); /*[out]*/

/*********************************************************************/
/**
 * psk_idx_linspace
 *  Build a vector with a given start and stop value and defined step size.
 *  This function uses integer values, so the resulting vector can be used
 *  as a indexing vector for signals.
 *  @param start The value of the first element
 *  @param step The step size. Difference between two consecutive elements
 *  @param stop The value of the last element
 *  @param len Pointer where to save the length of the vector
 *  @return The resulting vector or NULL, if start >= stop
 */
psk_uint32* psk_idx_linspace( psk_uint32 start,  /*[in]*/
                              psk_uint32 step,   /*[in]*/
                              psk_uint32 stop,   /*[in]*/
                              psk_uint32* len ); /*[out]*/

#endif //PSK_MATH_H

/*********************************************************************/
/*** psk_math.c                                                    ***/
/***  DESCIRPTION: Implementation of psk_math.h                    ***/
/***  Contains the function implementation of used mathematical    ***/
/***  functions for the VHBR PSK wave shape analysis tool          ***/
/*********************************************************************/

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

#include "psk_defines.h"
#include "psk_types.h"
#include "psk_math.h"

//-------------------------------------------------------------------
psk_double psk_mean( psk_double* vec, psk_uint32 len )
```

```
{
  psk_double sum = 0.0;
  psk_uint32 idx;

  if( len < 2 )
    return sum;

  for( idx = 0; idx < len; idx++ )
    sum += vec[idx];

  sum /= (double)len;

  return( sum );
}

//---------------------------------------------------------------------------
psk_complex psk_cmpl_mean( psk_complex* vec, psk_uint32 len )
{
  psk_complex mean;
  RE( mean ) = 0.0;
  IM( mean ) = 0.0;
  psk_uint32 idx;

  if( len < 2 )
    return mean;

  for( idx = 0; idx < len; idx++ )
  {
    RE( mean ) += RE( vec[idx] );
    IM( mean ) += IM( vec[idx] );
  }

  RE( mean ) /= (double)len;
  IM( mean ) /= (double)len;

  return( mean );
}

//---------------------------------------------------------------------------
psk_double* psk_diff( psk_double* vec , psk_uint32 len )
{
  psk_double* diff = calloc( ( len - 1 ), sizeof(psk_double) );
  psk_uint32 idx;

  for( idx = 0; idx < ( len - 1 ); idx++ )
    diff[idx] = vec[idx + 1] - vec[idx];

  return( diff );
}

//---------------------------------------------------------------------------
void psk_max( psk_double* vec,
              psk_uint32 vec_len,
              psk_double* max_val,
              psk_uint32* max_idx )
{
  if( vec == NULL )
    return;
  psk_uint32 idx;
  *max_val = vec[0];
  *max_idx = 0;

  for( idx = 1; idx < vec_len; idx++ )
  {
    if( vec[idx] > *max_val )
    {
      *max_val = vec[idx];
      *max_idx = idx;
    }
  }
}
```

```
//-----------------------------------------------------------------------
void psk_min( psk_double* vec,
              psk_uint32 vec_len,
              psk_double* min_val,
              psk_uint32* min_idx )
{
  psk_uint32 idx;
  *min_val = vec[0];
  *min_idx = 0;

  for( idx = 1; idx < vec_len; idx++ )
  {
    if( vec[idx] < *min_val )
    {
      *min_val = vec[idx];
      *min_idx = idx;
    }
  }
}

//-----------------------------------------------------------------------
psk_complex psk_add( psk_complex a, psk_complex b )
{
  psk_complex sum;
  RE( sum ) = RE( a ) + RE( b );
  IM( sum ) = IM( a ) + IM( b );

  return( sum );
}

//-----------------------------------------------------------------------
psk_complex psk_sub( psk_complex a, psk_complex b )
{
  psk_complex diff;
  RE( diff ) = RE( a ) - RE( b );
  IM( diff ) = IM( a ) - IM( b );

  return( diff );
}

//-----------------------------------------------------------------------
psk_complex psk_cmpl_mult( psk_complex a, psk_complex b )
{
  psk_complex prod;
  RE( prod ) = RE( a ) * RE( b ) - IM( a ) * IM( b );
  IM( prod ) = IM( a ) * RE( b ) + RE( a ) * IM( b );

  return( prod );
}

//-----------------------------------------------------------------------
psk_complex psk_cmpl_div( psk_complex a, psk_complex b )
{
  psk_complex quot;
  RE( quot ) = RE( a ) * RE( b ) + IM( a ) * IM( b );
  RE( quot ) = RE( quot ) / ( RE( b ) * RE( b ) + IM( b ) * IM( b ) );

  IM( quot ) = IM( a ) * RE( b ) - RE( a ) * IM( b );
  IM( quot ) = IM( quot ) / ( RE( b ) * RE( b ) + IM( b ) * IM( b ) );
  return( quot );
}

//-----------------------------------------------------------------------
psk_complex psk_cmpl_conj( psk_complex a )
{
  psk_complex conj;
  RE( conj ) = RE( a );
  IM( conj ) = -IM( a );

  return( conj );
```

```
}

//-------------------------------------------------------------------------
psk_double psk_abs( psk_complex num )
{
  return( sqrt( pow( RE( num ), 2 ) + pow( IM( num ), 2 ) ) );
}

//-------------------------------------------------------------------------
psk_complex* psk_cmpl_vec_mult( psk_complex* a,
                                psk_complex* b,
                                psk_uint32 len )
{
  psk_complex *prod_vec = calloc( len, sizeof(psk_complex) );
  psk_uint32 idx;

  for( idx = 0; idx < len; idx++ )
    prod_vec[idx] = psk_cmpl_mult( a[idx], b[idx] );

  return( prod_vec );
}

//-------------------------------------------------------------------------
psk_double* psk_vec_abs( psk_complex* a, psk_uint32 len )
{
  psk_double *abs_vec = calloc( len, sizeof(psk_complex) );
  psk_uint32 idx;

  for( idx = 0; idx < len; idx++ )
    abs_vec[idx] = psk_abs( a[idx] );

  return( abs_vec );
}

//-------------------------------------------------------------------------
psk_complex* psk_cmpl_vec_conj( psk_complex* vec, psk_uint32 len )
{
  psk_complex *conj_vec = calloc( len, sizeof(psk_complex) );
  psk_uint32 idx;

  for( idx = 0; idx < len; idx++ )
    conj_vec[idx] = psk_cmpl_conj( vec[idx] );

  return( conj_vec );
}

//-------------------------------------------------------------------------
psk_double* psk_vec_angle( psk_complex* vec, psk_uint32 len)
{
  psk_double *angle_vec = calloc( len, sizeof(psk_complex) );
  psk_uint32 idx;

  for( idx = 0; idx < len; idx++ )
    angle_vec[idx] = atan2( IM( vec[idx] ), RE( vec[idx] ) );

  return( angle_vec );
}

//-------------------------------------------------------------------------
psk_double psk_variance( psk_double* vec, psk_uint32 len )
{
  psk_double variance = 0.0;
  psk_double mean = psk_mean( vec, len );
  psk_uint32 idx;

  for( idx = 0; idx < len; idx++ )
    variance += pow( vec[idx] - mean, 2.0 );
  variance /= (double)len;

  return( variance );
}
```

```
//-----------------------------------------------------------------------
psk_double psk_std( psk_double* vec, psk_uint32 len )
{
  psk_double std_deviation = 0.0;
  std_deviation = psk_variance( vec, len );
  return( sqrt( std_deviation ) );
}

//-----------------------------------------------------------------------
psk_double* psk_normalize( psk_double* vec, psk_uint32 len )
{
  psk_double max_val = ABS( vec[0] );
  psk_double* norm_vec = calloc( len, sizeof(psk_double) );
  psk_uint32 idx;

  for( idx = 0; idx < len; idx++ )
  {
    if ( ABS( vec[idx] ) > max_val )
      max_val = ABS( vec[idx] );
  }

  for( idx = 0; idx < len; idx++ )
    norm_vec[idx] = vec[idx] / max_val;

  return( norm_vec );
}

//-----------------------------------------------------------------------
psk_complex* psk_cmpl_normalize( psk_complex* vec, psk_uint32 len )
{
  psk_double max_abs = psk_abs( vec[0] );
  psk_complex* norm_vec = calloc( len, sizeof(psk_complex) );
  psk_uint32 idx;
  psk_double tmp = 0.0;

  // find absolute maxima
  for( idx = 1; idx < len; idx++ )
  {
    tmp = psk_abs( vec[idx] );
    if( tmp > max_abs )
      max_abs = tmp;
  }

  // normalize
  for( idx = 0; idx < len; idx++ )
  {
    RE( norm_vec[idx] ) = RE( vec[idx] ) / max_abs;
    IM( norm_vec[idx] ) = IM( vec[idx] ) / max_abs;
  }

  return( norm_vec );
}

//-----------------------------------------------------------------------
psk_double* psk_linspace( psk_double start,
                          psk_double step,
                          psk_double stop,
                          psk_uint32* len )
{
  *len = ABS( ( (stop - start ) / step ) ) + 1.0;
  if( *len < 2 )
    return( NULL );
  psk_double* out = calloc( *len, sizeof(psk_double) );
  psk_uint32 idx = 0;

  out[idx] = start;
  for( idx = 1; idx < *len; idx++ )
    out[idx] = out[idx - 1] + step;

  return( out );
```

```
}

//-------------------------------------------------------------------------
psk_uint32* psk_idx_linspace( psk_uint32 start,
                              psk_uint32 step,
                              psk_uint32 stop,
                              psk_uint32* len )
{
  if( start >= stop || step == 0 )
    return( NULL );
  *len = ( ( ( (psk_double)stop - (psk_double)start ) /
           (psk_double)step ) + 1.0 );
  psk_uint32 *out = calloc( *len, sizeof(psk_uint32) );
  psk_uint32 idx = 0;

  out[idx] = start;
  for( idx = 1; idx < *len; idx++ )
    out[idx] = out[idx - 1] + step;

  return( out );
}

/*************************************************************************/
/*** psk_dsp.h                                                         ***/
/***  DESCIRPTION: header of psk_dsp.c                                 ***/
/***    Collection of functions used for dsp in the PSK waveform       ***/
/***    characteristics analysis tool.                                 ***/
/*************************************************************************/

#ifndef PSK_DSP_H
#define PSK_DSP_H

#include "psk_types.h"
#include "psk_math.h"

/*************************************************************************/
/**
 * psk_compute_butter_lp_coef
 *  Computes the low-pass filter coefficients of a 4th order Butterworth
 *  IIR filter with a cut-off frequency of 120MHz
 *  @param sample_rate Sampling rate
 *  @return Struct with the b and a polynom coefficients
 */
psk_butter_coefs psk_compute_butter_lp_coef( psk_double sample_rate /*[in]*/);

/*************************************************************************/
/**
 * psk_antialiasing filter
 *  Filters the signal with a 4th order IIR filter. This function makes use of
 *  #psk_compute_butter_lp_coef to get the filter coefficients.
 *  @param v_in Input signal amplitude
 *  @param len_in Length of v_in
 *  @param v_out_ptr This pointer will be set to the array containing the
 *         output signal.
 *  @param len_out Length of the output signal
 *  @param sample_rate Sampling rate of the input signal
 *  @return error value as defined in psk_defines.h
 */
psk_int32 psk_antialiasing( psk_double* v_in,         /*[in]*/
                            psk_uint32 len_in,        /*[in]*/
                            psk_double** v_out_ptr,   /*[out]*/
                            psk_uint32* len_out,      /*[out]*/
                            psk_double sample_rate ); /*[in]*/

/*************************************************************************/
/**
 * psk_downsample
 *  resamples input signal to internal sampling rate (FS_INT)
 *  defined in "psk_defines.h" using linear interpolation
 *  @param samples_in Array of amplitude values
 *  @param time_in Array of time values
```

```
 *  @param len_in Length of samples_in and time_in
 *  @param samples_out_ptr Pointer to output array of amplitudes
 *  @param time_out_ptr Pointer to output array of time values
 *  @param len_out Length of output arrays
 *  @return error value as defined in psk_defines.h
 */
psk_int32 psk_downsample( psk_double* samples_in       /*[in]*/,
                          psk_double* time_in          /*[in]*/,
                          psk_uint32 len_in            /*[in]*/,
                          psk_double** samples_out_ptr /*[out]*/,
                          psk_double** time_out_ptr    /*[out]*/,
                          psk_uint32* len_out          /*[out]*/ );

/*****************************************************************************/
/**
 * psk_ma_filter
 *  Moving average implementation
 *  @param samples_in Array of amplitude values
 *  @param len_in Length of samples_in
 *  @param samples_out_ptr Pointer to output array of amplitudes
 *  @param len_out Length of output array
 *  @param len_filter Window size of the moving average filter
 *  @return error value as defined in psk_defines.h
 */
psk_int32 psk_ma_filter( psk_double* samples_in        /*[in]*/,
                         psk_uint32 len_in             /*[in]*/,
                         psk_double** samples_out_ptr  /*[out]*/,
                         psk_uint32* len_out           /*[out]*/,
                         psk_uint32 len_filter         /*[in]*/ );

/*****************************************************************************/
/**
 * psk_ma_filter_cmpl
 *  Moving average on a complex input signal: real and imaginary part are
 *  each MA filtered
 *  @param samples_in Array of amplitude values
 *  @param len_in Length of samples_in
 *  @param samples_out_ptr Pointer to output array of amplitudes
 *  @param len_out Length of output array
 *  @param len_filter Window size of the moving average filter
 *  @return error value as defined in psk_defines.h
 */
psk_int32 psk_ma_filter_cmpl( psk_complex* samples_in        /*[in]*/,
                              psk_uint32 len_in              /*[in]*/,
                              psk_complex** samples_out_ptr  /*[out]*/,
                              psk_uint32* len_out            /*[out]*/,
                              psk_uint32 len_filter          /*[in]*/);

/*****************************************************************************/
/**
 * psk_FIR_filter
 *  finite impulse response filter
 *  @param b Polynomial ff filter coefficients "Direct Form II Transposed"
 *  @param num_coef Number of coefficients
 *  @param sample_in Array of amplitude values
 *  @param len_in Length of sample_in
 *  @return filter signals of the same length as input signal
 */
psk_double* psk_FIR_filter( psk_double* b          /*[in]*/,
                            psk_uint32 num_coef    /*[in]*/,
                            psk_double* sample_in  /*[in]*/,
                            psk_uint32 len_in      /*[in]*/);

/*****************************************************************************/
/**
 * psk_FIR_filter_cmpl
 *  complex finite impulse response filter "Direct Form II Transposed"
 *  @param b Complex valued polynomial ff filter coefficients
 *  @param num_coef Number of coefficients
 *  @param sample_in Complex samples_in (arra of amplitude values)
 *  @param len_in Length of samples_
```

```
 *  @return complex filter signals of the same length as input signal
 */
psk_complex* psk_FIR_filter_cmpl( psk_complex* b          /*[in]*/,
                                  psk_uint32 num_coef     /*[in]*/,
                                  psk_complex* sample_in  /*[in]*/,
                                  psk_uint32 len_in       /*[in]*/);

/*****************************************************************************/
/**
 * psk_demodulation
 *  Homodyne demodulator (IQ demodulator)
 *  @param samples_in Array of amplitude values
 *  @param time_in Array of time values
 *  @param len Length of inputs
 *  @param v_out_bb complex output signal in baseband of size len
 *  @return nothing
 */
void psk_demodulation( psk_double*   samples_in /*[in]*/,
                       psk_double*   time_in    /*[in]*/,
                       psk_uint32    len        /*[in]*/,
                       psk_complex** v_out_bb   /*[out]*/);

/*****************************************************************************/
/**
 * psk_exp_1i_wt
 *  complex multiplication with exp(j *w*t) term
 *  @param v_in_bb Array of amplitude values
 *  @param time_in Array of time values
 *  @param len Length of inputs
 *  @param freq Frequency parameter for computing 'w'
 *  @param v_out_bb complex output signal in baseband
 *         (memory not allocated in function)
 *  @return nothing
 */
void psk_exp_1i_wt( psk_complex* v_in_bb    /*[in]*/,
                    psk_double*  time_in    /*[in]*/,
                    psk_uint32   len        /*[in]*/,
                    psk_double   freq       /*[in]*/,
                    psk_complex** v_out_bb  /*[out]*/);

/*****************************************************************************/
/**
 * psk_derotation
 *  derotate input baseband signal by frequency f
 *  @param v_in_bb Complex valued array of amplitude values
 *  @param time_in Array of time values of v_in_bb
 *  @param len Length of input arrays
 *  @param v_out_bb complex output signal in baseband
 *         (memory not allocated in function)
 *  @return error value as defined in psk_defines.h
 */
psk_int32 psk_derotation( psk_complex* v_in_bb    /*[in]*/,
                          psk_double* time_in     /*[in]*/,
                          psk_uint32  len         /*[in]*/,
                          psk_complex** v_out_bb  /*[out]*/);

/*****************************************************************************/
/**
 * psk_estimate_carrier_freq
 *  estimate frequency difference of input signal to demodulation signal
 *  @param v_in_bb (complex valued array)
 *  @param len_unmod Length of unmodulated signal
 *  @return frequency error [Hz]
 */
psk_double psk_estimate_carrier_freq( psk_complex* v_in_bb  /*[in]*/,
                                      psk_uint32 len_unmod  /*[in]*/);

/*****************************************************************************/
/**
 * psk_cross_coveriance
 *  resample signal by rate 'osf' using nearest neighbor interpolation
```

```
 *  @param v_in1 (complex valued array)
 *  @param len_in1 Length of v_in1
 *  @param v_in2 (complex valued array)
 *  @param len_in2 Length v_in2
 *  @param lags The signals are shifted 'lags' times
 *  @param v_out Complex valued array containing the xcov function of v_in1 and
 *         v_in2
 *  @param lagIdx_vec Array of lag indices
 *  @return error value as defined in psk_defines.h
 */
psk_int32 psk_cross_covariance( psk_complex* v_in1       /*[in]*/,
                                psk_uint32   len_in1     /*[in]*/,
                                psk_complex* v_in2       /*[in]*/,
                                psk_uint32   len_in2     /*[in]*/,
                                psk_uint32   lags        /*[in]*/,
                                psk_complex** v_out      /*[out]*/,
                                psk_int32**   lagIdx_vec /*[out]*/);

/*************************************************************************/
/**
 * psk_ff_lms
 *  System identification using LMS algorithm
 *  @param v_in1 (complex valued array ==> reference data (send data))
 *  @param v_in2 (complex valued array ==> filtered data (received data)
 *  @param len Length v_in1, v_in2
 *  @param n_taps (filter length used for SI)
 *  @return complex filter coefficients
 */
 psk_complex* psk_ff_lms( psk_complex* v_in1  /*[in]*/,
                          psk_complex* v_in2  /*[in]*/,
                          psk_uint32 len      /*[in]*/,
                          psk_uint32 n_taps   /*[in]*/);

/*************************************************************************/
/**
 * psk_cross_coveriance
 *  resample signal by rate 'osf' using nearest neighbor interpolation
 *  @param v_in (input phase array in radians)
 *  @param len_in Length of v_in
 *  @return v_out Unwrapped output phase array in radians
 */
psk_double* psk_unwrap(psk_double* v_in /*[in]*/, psk_uint32 len_in /*[in]*/);

/*************************************************************************/
/**
 * psk_get_phase_noise
 *  calculate the differential phase noise of a given complex signal
 *  @param sig Complex signal of an unmodulated carrier sequence
 *  @param sig_len The number of samples in sig
 *  @param phase_noise Pointer where to store the result
 *  @return error value as defined in psk_defines.h
 */
psk_int32 psk_get_phase_noise( psk_complex* sig,          /*[in]*/
                               psk_uint32 sig_len,        /*[in]*/
                               psk_double* phase_noise ); /*[out]*/

/*************************************************************************/
/**
 * psk_isi_param
 *  calculate the differential phase noise of a given complex signal
 *  @param sig_bb_int Grid aligned complex input BB signal
 *  @param sig_bb_len The number of samples in sig_bb_int
 *  @param isi_m Pointer where the ISI magnitude [EPI] will be stored
 *  @param isi_d Pointer where the ISI rotation [degree] will be stored
 *  @param phase_range Pointer where the measured phase range will be stored
 *  @param sig_out Pointer where the grid aligned signal at symbol rate will
 *                 be stored
 *  @return error value as defined in psk_defines.h
 */
psk_int32 psk_isi_param( psk_complex* sig_bb_int,  /*[in]*/
                         psk_uint32 sig_bb_len,    /*[in]*/
```

```
                        psk_double* isi_m,        /*[out]*/
                        psk_double* isi_d,        /*[out]*/
                        psk_double* phase_range,  /*[out]*/
                        psk_complex** sig_out );  /*[out]*/


 /*********************************************************************/
/**
 * psk_re_align_symbol_grid
 *  align the PSK modulated input signal to the reference SOC signal
 *  @param v_in Complex base band signal
 *  @param len The number of samples in v_in
 *  @param strt_idx Pointer to field storing the index of the starting point
 *  @param end_idx Pointer to field storing the index of the end point
 *  @return error value as defined in psk_defines.h
 */
 psk_int32 psk_re_align_symbol_grid( psk_complex* v_in,       /*[in]*/
                                     psk_uint32 len,          /*[in]*/
                                     psk_uint32* strt_idx,    /*[out]*/
                                     psk_uint32* end_idx );   /*[out]*/

 /*********************************************************************/
/**
 * psk_find_approx_delay
 *  align the PSK modulated input signal to the reference SOC signal
 *  @param v_ref_in Complex reference base band signal
 *  @param ref_len The number of samples in v_ref_in
 *  @param v_in Complex base band signal
 *  @param len The number of samples in v_in
 *  @param max_shift Maximum shift used for cross-covaraince computation
  * @param delay
 *  @return error code as defined in psk_defines.h
 */
 psk_int32 psk_find_approx_delay( psk_complex* v_ref_in,        /*[in]*/
                                  psk_uint32 ref_len,           /*[in]*/
                                  psk_complex* v_in,            /*[in]*/
                                  psk_uint32 len,               /*[in]*/
                                  psk_uint32 max_shift,         /*[in]*/
                                  psk_uint32 *delay );          /*[out]*/

/*********************************************************************/
#endif // PSK_DSP_H

/*********************************************************************/
/*** psk_dsp.c                                                     ***/
/***  DESCIRPTION: Implementation of psk_dsp.h                     ***/
/***  Contains the function implementation of used DSP functions   ***/
/***  for the VHBR PSK wave shape analysis tool                    ***/
/*********************************************************************/
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>

#include "psk_defines.h"
#include "psk_types.h"
#include "psk_math.h"
#include "psk_dsp.h"

//---------------------------------------------------------------------
psk_butter_coefs psk_compute_butter_lp_coef( psk_double sample_rate )
{
  psk_uint32 i = 0;
  psk_uint32 butter_order = ( BUTTER_SIZE_A - 1 );
  psk_double wd = 120e6 * 2*M_PI; // cutoff frequency in radians ==> 120 Mhz
  psk_double Ts = 1.0 / (psk_double)sample_rate;
  psk_double wa = 0.0;
  psk_double wa_p2 = 0.0;
  psk_double g = 0.0;
  psk_double a_biquad[3 * butter_order / 2];
  psk_double b_biquad[3 * butter_order / 2 ];
```

```
  psk_double norm_fact = 0.0;

  psk_butter_coefs coefs;

  memset( a_biquad, 0, 3 * butter_order / 2 );
  memset( b_biquad, 0, 3 * butter_order / 2 );

  if( butter_order == 0 ) // filter order must be greater then Zero and even
    return( coefs );

  if( butter_order % 2 ) // filter order must be even
    return(coefs);

  // Step 1: pre-warping of cut-off frequency
  wa = tan( wd * Ts / 2.0 );
  //wa1 = 2.0 / Ts * wa;

  wa_p2 = pow( wa, 2 );
  // Step 2: compute biquad filter coefficients

  for( i = 0 ; i < ( butter_order / 2 ); i++ )
  {
    norm_fact = 1.0 + 2.0 * cos( M_PI * ( 2.0 * (psk_double)i + 1.0 ) /
                ( 2.0 * (psk_double)butter_order ) ) * wa + wa_p2 ;
    g = wa_p2 / norm_fact;

    b_biquad[i * 3    ] = 1.0 * g;
    b_biquad[i * 3 + 1] = 2.0 * g;
    b_biquad[i * 3 + 2] = 1.0 * g;

    a_biquad[i * 3    ] = 1.0;
    a_biquad[i * 3 + 1] = ( 2.0 * wa_p2 - 2.0 ) / norm_fact;
    a_biquad[i * 3 + 2] = ( 1.0 - 2.0 * cos( M_PI * ( 2.0 * (psk_double)i + 1.0 ) /
              ( 2.0 * (psk_double)butter_order ) ) * wa + wa_p2 ) / norm_fact;
  }

  // compute polynomial from 2 biquads (SOS)
  if( butter_order == 4 )
  {
    //(a0*s^2+a1*s+a2)*(b0*s^2+b1*s+b2)= s^4(a0*b0)+s^3(a1*b0+a0*b1)+s^2(a2*b0+a1*b1+a0*b2
)+s(a2*b1+a1*b2)+a2*b2
    coefs.b[0] = b_biquad[0] * b_biquad[3];
    coefs.b[1] = b_biquad[1] * b_biquad[3] + b_biquad[0] * b_biquad[4];
    coefs.b[2] = b_biquad[2] * b_biquad[3] + b_biquad[1] * b_biquad[4] + b_biquad[0] * b_
biquad[5];
    coefs.b[3] = b_biquad[2] * b_biquad[4] + b_biquad[1] * b_biquad[5];
    coefs.b[4] = b_biquad[2] * b_biquad[5];

    coefs.a[0] = a_biquad[0] * a_biquad[3];
    coefs.a[1] = a_biquad[1] * a_biquad[3] + a_biquad[0] * a_biquad[4];
    coefs.a[2] = a_biquad[2] * a_biquad[3] + a_biquad[1] * a_biquad[4] + a_biquad[0] * a_
biquad[5];
    coefs.a[3] = a_biquad[2] * a_biquad[4] + a_biquad[1] * a_biquad[5];
    coefs.a[4] = a_biquad[2] * a_biquad[5];

  }

  return( coefs );
}

//----------------------------------------------------------------------------
psk_int32 psk_antialiasing( psk_double*  v_in,
                            psk_uint32   len_in,
                            psk_double** v_out_ptr,
                            psk_uint32*  len_out,
                            psk_double   sample_rate )
{

  psk_double len_butter_out = (psk_double)len_in - (psk_double)BUTTER_SIZE_A;
  psk_double* v_out = calloc( sizeof(psk_double), len_butter_out );
```

```
  if( v_out == NULL )
    return( PSK_ERR_OUT_OF_MEM );

  psk_double* out_tmp = calloc( sizeof(psk_double), len_in );
  if( out_tmp == NULL )
  {
    if( v_out )
      free( v_out );

    return( PSK_ERR_OUT_OF_MEM );
  }

  psk_int32 i = 0;
  psk_uint32 t_0 = 0;
  psk_uint32 t_1 = 0;
  psk_uint32 t_2 = 0;
  psk_uint32 t_3 = 0;
  psk_uint32 t_4 = 0;
  psk_butter_coefs coefs;


  coefs = psk_compute_butter_lp_coef( sample_rate );
  for( i = 0; i < len_in; i++ )
  {
    if( i < BUTTER_SIZE_A - 1 )
    {
      t_0 = fmax( 0, i     );
      t_1 = fmax( 0, i - 1 );
      t_2 = fmax( 0, i - 2 );
      t_3 = fmax( 0, i - 3 );
      t_4 = fmax( 0, i - 4 );
      out_tmp[i] = coefs.b[0] * v_in[t_0] + coefs.b[1] * v_in[t_1] +
                   coefs.b[2] * v_in[t_2] + coefs.b[3] * v_in[t_3] +
                   coefs.b[4] * v_in[t_4] -
                   ( coefs.a[0] * out_tmp[t_0] + coefs.a[1] * out_tmp[t_1] +
                   coefs.a[2] * out_tmp[t_2] + coefs.a[3] * out_tmp[t_3] +
                   coefs.a[4] * out_tmp[t_4] );
    }
    else
    {
      out_tmp[i] = coefs.b[0] * v_in[i] + coefs.b[1] * v_in[i - 1] +
                   coefs.b[2] * v_in[i - 2] + coefs.b[3] * v_in[i - 3] +
                   coefs.b[4] * v_in[i - 4] -
                   ( coefs.a[0] * out_tmp[i] + coefs.a[1] * out_tmp[i - 1] +
                   coefs.a[2] * out_tmp[i - 2] + coefs.a[3] * out_tmp[i - 3] +
                   coefs.a[4] * out_tmp[i - 4] );
    }
    if( i > BUTTER_SIZE_A - 1 )
    {
      v_out[i - BUTTER_SIZE_A] = out_tmp[i];
    }
  }

  *v_out_ptr = v_out;
  *len_out = len_butter_out;

  if( out_tmp )
    free( out_tmp );
  out_tmp = NULL;

  return( PSK_ERR_OK );
}

//---------------------------------------------------------------------
psk_int32 psk_downsample( psk_double*  samples_in,
                          psk_double*  time_in,
                          psk_uint32   len_in,
                          psk_double** samples_out_ptr,
                          psk_double** time_out_ptr,
                          psk_uint32*  len_out)
{
```

```
  psk_uint32 k, i; // indexing counters
  psk_double d_y, d_t1, d_t2;
  k = i = 1;
  d_y = d_t1 = d_t2 = 0;
  psk_double t_step = 1.0 / (psk_double)FS_INT;

  psk_double* t_out  = NULL;
  psk_uint32 vec_size;
  t_out = psk_linspace( time_in[0], t_step, time_in[len_in - 1], &vec_size );

  psk_double* v_out = calloc( vec_size, sizeof(psk_double) );
  if( v_out == NULL )
    return( PSK_ERR_OUT_OF_MEM );

  v_out[0] = samples_in[0];

  for( i = 1; i < vec_size; i++)
  {
    k--;
    while ( t_out[i] > time_in[i+k] )
    {
      k++;
    }
    d_y = samples_in[ i + k ] - samples_in[ i + k - 1 ];
    d_t1 = time_in[ i + k ] - time_in[ i + k - 1 ];
    d_t2 = t_out[ i ] - time_in[ i + k - 1 ];
    v_out[i] = samples_in[ i + k - 1 ] + d_y * ( d_t2 / d_t1 );
  }

  *samples_out_ptr = v_out;
  *time_out_ptr = t_out;
  *len_out = vec_size;

  return( PSK_ERR_OK );
}

// ----------------------------------------------------------------------------
psk_int32 psk_ma_filter( psk_double*  samples_in,
                         psk_uint32   len_in,
                         psk_double** samples_out_ptr,
                         psk_uint32*  len_out,
                         psk_uint32   len_filter )
{
  psk_double out_sum, out_sum_prev, a_prev;
  psk_uint32 i, k;
  out_sum = out_sum_prev = a_prev = 0;
  psk_double* filt_buffer = NULL;
  psk_uint32 N_out = len_in - len_filter;
  if( N_out < 1 )
    return( PSK_ERR_PARAMETER );

  i = k = 0;

  filt_buffer = calloc( len_filter, sizeof(psk_double) );
  if( filt_buffer == NULL )
    return( PSK_ERR_OUT_OF_MEM );

  psk_double* v_out = calloc( N_out , sizeof(psk_double) );
  if( v_out == NULL )
  {
    free( filt_buffer );
    return( PSK_ERR_OUT_OF_MEM );
  }

  for( i = 0; i < len_in; i++ )
  {
    a_prev = filt_buffer[len_filter - 1];

    for( k = len_filter - 1; k > 0; k-- )
      filt_buffer[k] = filt_buffer[k - 1];
```

```
    filt_buffer[0] = samples_in[i];
    out_sum_prev = filt_buffer[0] - a_prev;
    out_sum += out_sum_prev;

    if( i > len_filter - 1 )
      v_out[i - len_filter] = out_sum / (psk_double)len_filter;
  }

  if( filt_buffer )
    free( filt_buffer );

  *samples_out_ptr = v_out;
  *len_out = N_out;
  return( PSK_ERR_OK );
}

//-----------------------------------------------------------------------------
psk_int32 psk_ma_filter_cmpl( psk_complex*  samples_in,
                              psk_uint32    len_in,
                              psk_complex** samples_out_ptr,
                              psk_uint32*   len_out,
                              psk_uint32    len_filter )
{
  psk_complex out_sum, out_sum_prev, a_prev;
  psk_uint32 i, k;
  RE( out_sum ) = RE( out_sum_prev ) = RE( a_prev ) = 0.0;
  IM( out_sum ) = IM( out_sum_prev ) = IM( a_prev ) = 0.0;
  psk_complex* filt_buffer = NULL;
  psk_uint32 N_out = len_in - len_filter;
  if( N_out < 1 )
    return( PSK_ERR_PARAMETER );

  i = k = 0;

  filt_buffer = calloc( len_filter, sizeof(psk_complex) );
  if( filt_buffer == NULL )
    return( PSK_ERR_OUT_OF_MEM );

  psk_complex* v_out = calloc( N_out , sizeof(psk_complex) );
  if( v_out == NULL )
  {
    free( filt_buffer );
    return( PSK_ERR_OUT_OF_MEM );
  }

  for( i = 0; i < len_in; i++ )
  {
    a_prev = filt_buffer[len_filter - 1];
    for( k = len_filter - 1; k > 0; k-- )
      filt_buffer[k] = filt_buffer[k - 1];

    filt_buffer[0] = samples_in[i];
    out_sum_prev = psk_sub( filt_buffer[0] , a_prev );
    out_sum = psk_add( out_sum, out_sum_prev );

    if( i > len_filter - 1 )
    {
      RE( v_out[i - len_filter] ) = RE( out_sum ) / (psk_double)len_filter;
      IM( v_out[i - len_filter] ) = IM( out_sum ) / (psk_double)len_filter;
    }
  }

  if( filt_buffer )
    free( filt_buffer );

  *samples_out_ptr = v_out;
  *len_out = N_out;

  return( PSK_ERR_OK );
}
```

```
//-----------------------------------------------------------------------
psk_complex* psk_FIR_filter_cmpl( psk_complex* b,
                                  psk_uint32   num_coef,
                                  psk_complex* sample_in,
                                  psk_uint32   len_in )

{
  psk_uint32 i, k;
  psk_uint32 idx_ptr, idx;
  i = k = idx_ptr = idx = 0;
  psk_complex* circ_buffer = calloc( num_coef, sizeof(psk_complex) );
  psk_complex* out         = calloc( len_in,   sizeof(psk_complex) );
  psk_complex y, res_mult;

  for( i = 0; i < len_in; i++ )
  {
    circ_buffer[idx_ptr] = sample_in[i];

    idx_ptr = ( (idx_ptr + 1) % num_coef);
    RE(y) = 0.0;
    IM(y) = 0.0;
    for( k = 0; k < num_coef; k++ )
    {
      idx = ( ( idx_ptr + k ) % num_coef );
      res_mult = psk_cmpl_mult( b[num_coef - k - 1], circ_buffer[idx] );
      y = psk_add( y, res_mult );
    }
    out[i] = y;
  }

  if( circ_buffer )
    free( circ_buffer );

  return( out );
}

//-----------------------------------------------------------------------
psk_double* psk_FIR_filter( psk_double* b,
                            psk_uint32  num_coef,
                            psk_double* sample_in,
                            psk_uint32  len_in )
{
  psk_uint32 i, k;
  psk_uint32 idx_ptr, idx;
  i = k = idx_ptr = idx = 0;
  psk_double* circ_buffer = calloc( num_coef, sizeof(psk_double) );
  psk_double* out         = calloc(   len_in, sizeof(psk_double) );

  psk_double y;

  for( i = 0; i < len_in; i++ )
  {
    circ_buffer[idx_ptr] = sample_in[i];
    idx_ptr = ( ( idx_ptr + 1 ) % num_coef );
    y = 0.0;
    for (k = 0; k < num_coef; k++)
    {
      idx = ( ( idx_ptr + k ) % num_coef );
      y += ( b[num_coef - k - 1] * circ_buffer[idx] );
    }
    out[i] = y;
  }

  if( circ_buffer )
    free( circ_buffer );

  return( out );
}

//-----------------------------------------------------------------------
void psk_demodulation( psk_double*   samples_in,
```

```
                        psk_double*   time_in,
                        psk_uint32    len,
                        psk_complex** v_out_bb )
{
  psk_uint32 i = 0;
  psk_double w_c = 2.0 * (psk_double)M_PI * FC;
  psk_complex* v_out_bb1 = calloc( len, sizeof(psk_complex) );

  for( i = 0; i < len; i++ )
  {
    RE( v_out_bb1[i] ) = samples_in[i] * cos( w_c * time_in[i] );
    IM( v_out_bb1[i] ) = -1.0 * samples_in[i] * sin( w_c * time_in[i] );
  }
  *v_out_bb = v_out_bb1;
}

//---------------------------------------------------------------------------
void psk_exp_1i_wt( psk_complex*  v_in_bb,
                    psk_double*   time_in,
                    psk_uint32    len,
                    psk_double    freq,
                    psk_complex** v_out_bb )
{
  psk_complex* out_bb = calloc( len, sizeof(psk_complex) );
  psk_uint32 idx = 0;
  psk_double w_c = 2.0 * (psk_double)M_PI * freq;
  psk_complex arg2;

  for( idx = 0; idx < len; idx++ )
  {
    RE( arg2 ) = cosl( w_c * time_in[idx] );
    IM( arg2 ) = sinl( w_c * time_in[idx] );
    out_bb[idx] = psk_cmpl_mult( v_in_bb[idx], arg2 );

  }
  *v_out_bb = out_bb;
}

//---------------------------------------------------------------------------
psk_int32 psk_derotation( psk_complex*  v_in_bb,
                          psk_double*   time_in,
                          psk_uint32    len,
                          psk_complex** v_out_bb )
{
  psk_complex* ma_out = NULL;
  psk_uint32 filter_len = 4.0 * (psk_double)FS_INT / (psk_double)FC;
  psk_uint32 ma_len = 0;
  psk_int32 status = 0;

  psk_double fc_err = 0.0;

  status = psk_ma_filter_cmpl( v_in_bb,
                               (psk_uint32)IDX_UNMOD,
                               &ma_out,
                               &ma_len,
                               filter_len );

  fc_err = psk_estimate_carrier_freq( ma_out, ma_len );

  fc_err *= ( -1.0 * (psk_double)FS_INT );
  psk_exp_1i_wt( v_in_bb, time_in, len, fc_err, v_out_bb );

  return( status );
}

//---------------------------------------------------------------------------
psk_double psk_estimate_carrier_freq( psk_complex* v_in_bb,
                                      psk_uint32   len_unmod )
{
  psk_complex* v_in_diff      = calloc( len_unmod - 1, sizeof(psk_complex) );
  psk_complex* v_derotated_bb = NULL;
```

```
psk_double* re_v_in_norm    = calloc( len_unmod, sizeof(psk_double) );
psk_double* im_v_in_norm    = calloc( len_unmod, sizeof(psk_double) );
psk_double* re_diff, *im_diff, *phase, *v_abs;
re_diff = im_diff = phase = v_abs = NULL;
psk_double *derotate_idx = NULL;

psk_uint32 idx, idx_len;
psk_double cmp_abs, mu_v_abs, FcEst, FcEst_2, phase_rot;

idx = idx_len = 0;
psk_uint32 mu1_end_idx= 0;
psk_double mu_phase_start, mu_phase_end;
mu_phase_start = mu_phase_end = 0.0;

//---------------------------------------------------------------------
// 1 Step: frequency estimate based on first derivative
// normalize to abs of each element
//---------------------------------------------------------------------

for( idx = 0; idx < len_unmod; idx++ )
{
  cmp_abs = psk_abs( v_in_bb[idx] );

  re_v_in_norm[idx] = RE( v_in_bb[idx] ) / cmp_abs;
  im_v_in_norm[idx] = IM( v_in_bb[idx] ) / cmp_abs;
}

// compute first difference
re_diff = psk_diff( re_v_in_norm, len_unmod );
im_diff = psk_diff( im_v_in_norm, len_unmod );

// form complex vector
for( idx = 0; idx < ( len_unmod - 1 ); idx++ )
{
  RE( v_in_diff[idx] ) = re_diff[idx];
  IM( v_in_diff[idx] ) = im_diff[idx];
}
// computs complex abs
v_abs = psk_vec_abs( v_in_diff, ( len_unmod - 1 ) );

mu_v_abs = psk_mean( v_abs, ( len_unmod - 1 ) );
// estimated frequency offset to FC
FcEst  = asinl( ( mu_v_abs / 2.0 ) ) / (psk_double)M_PI;

derotate_idx = psk_linspace( 1.0, 1.0, (psk_double)len_unmod, &idx_len );

// derotate signal for step 2 of frequency offset determination
psk_exp_1i_wt( v_in_bb, derotate_idx,
               len_unmod,
               (-1.0) * FcEst,
               &v_derotated_bb );

// compute phase
phase = psk_vec_angle( v_derotated_bb, len_unmod );
// unwrap phase
psk_double* uw_phase = psk_unwrap( phase, len_unmod );

// free not use memory

free(re_v_in_norm);   re_v_in_norm   = NULL;
free(im_v_in_norm);   im_v_in_norm   = NULL;
free(re_diff);        re_diff        = NULL;
free(im_diff);        im_diff        = NULL;
free(v_abs);          v_abs          = NULL;
free(v_derotated_bb); v_derotated_bb = NULL;
free(phase);

//---------------------------------------------------------------------
// Step 2: fine estimate of frequency difference to FC based on
// slope computation
//---------------------------------------------------------------------
```

```
  mu1_end_idx = len_unmod / 2;
  // compute mean of start and end section
  mu_phase_start = psk_mean( uw_phase, mu1_end_idx );

  for( idx = mu1_end_idx - 1; idx < len_unmod; idx++ )
    mu_phase_end += uw_phase[idx];

  mu_phase_end /= (psk_double)( len_unmod - mu1_end_idx + 1 );

  // compute slope and transform to frequency error with respect to FC
  phase_rot = mu_phase_end - mu_phase_start;

  FcEst_2 = phase_rot / 2.0 / (psk_double)M_PI /
            ( ( (psk_double)len_unmod - 1.0 ) / 2.0 );
  // complete frequency difference to FC
  FcEst += FcEst_2;

  free( uw_phase );
  phase = NULL;
  free( derotate_idx );
  derotate_idx = NULL;

  return( FcEst );
}

//-----------------------------------------------------------------------------
psk_int32 psk_cross_covariance( psk_complex*  v_in1,
                                psk_uint32    len_in1,
                                psk_complex*  v_in2,
                                psk_uint32    len_in2,
                                psk_uint32    lags,
                                psk_complex** v_out,
                                psk_int32**   lagIdx_vec )
{
  psk_complex mu_in1, mu_in2;
  psk_uint32 lagIdx_len, i;
  lagIdx_len = i = 0;

  psk_complex* v_in1_wo_mu = calloc( len_in1, sizeof(psk_complex) );
  psk_complex* v_in2_wo_mu = calloc( len_in2, sizeof(psk_complex) );

  // compute mean of input signal
  mu_in1 = psk_cmpl_mean( v_in1, len_in1 );
  mu_in2 = psk_cmpl_mean( v_in2, len_in2 );

  // subtract mean
  for( i = 0; i < len_in2; i++ )
    v_in2_wo_mu[i] = psk_sub( v_in2[i], mu_in2 );

  // compute conjugate complex
  v_in2_wo_mu = psk_cmpl_vec_conj( v_in2_wo_mu, len_in2 );

  // subtract mean
  if( lags == 0 )
  {
    psk_complex res1;
    psk_complex* out1 = calloc( 1, sizeof(psk_complex) );
    psk_int32* lagIdx1 = calloc( 1, sizeof(psk_int32) );
    *lagIdx1 = 0;
    RE(*out1) = 0.0;
    IM(*out1) = 0.0;
    for( i = 0; i < len_in1; i++ )
      v_in1_wo_mu[i] = psk_sub( v_in1[i], mu_in1 );

    for( i = 0; i< len_in1; i++ )
    {
      res1 = psk_cmpl_mult( v_in1_wo_mu[i], v_in2_wo_mu[i] );
      *out1 = psk_add( *out1, res1 );
    }

    *v_out = out1;
```

```
    *lagIdx_vec = lagIdx1;
  }
  else
  {
    psk_complex* out = NULL;
    psk_double* lagIdx = NULL;

    for( i = 0; i < len_in1; i++ )
      v_in1_wo_mu[len_in1 - ( 1 + i )] = psk_sub( v_in1[i], mu_in1 );

    out = psk_FIR_filter_cmpl( v_in1_wo_mu, len_in1, v_in2_wo_mu, lags );
    lagIdx = psk_linspace( (psk_double)( len_in1 - 1 ),
                           -1.0,
                           (psk_double)len_in1 - (psk_double)lags ,
                           &lagIdx_len );

    psk_int32* lagIdx_int32 = calloc(lagIdx_len, sizeof(psk_int32) );

    for( i = 0; i < lagIdx_len; i++ )
      lagIdx_int32[i] = (psk_int32)lagIdx[i];

    if( lagIdx_len != lags )
    {
      if( lagIdx )
        free( lagIdx );
      lagIdx = NULL;

      if( lagIdx_int32 )
        free( lagIdx_int32 );
      lagIdx_int32 = NULL;
      return( PSK_ERR_SIGNAL_LEN_MISMATCH );
    }

    *v_out = out;
    *lagIdx_vec = lagIdx_int32;

    if( lagIdx )
      free( lagIdx );
    lagIdx = NULL;
  }

  if( v_in1_wo_mu )
    free( v_in1_wo_mu );
  v_in1_wo_mu = NULL;

  if( v_in2_wo_mu )
    free( v_in2_wo_mu );
  v_in2_wo_mu = NULL;

  return( PSK_ERR_OK );
}

//---------------------------------------------------------------------------
psk_double* psk_unwrap( psk_double* v_in, psk_uint32 len_in )
{
  psk_uint32 idx, k_p, k_n;
  idx = k_p = k_n = 0;
  psk_double* out = calloc( len_in, sizeof(psk_double) );
  psk_double* in_diff = NULL;

  in_diff = psk_diff( v_in, len_in );
  out[0] = v_in[0];

  for( idx = 1; idx < len_in; idx++ )
  {
    if( in_diff[idx - 1] >= (psk_double)M_PI )
      k_p += 1;

    if( in_diff[idx - 1] <= ( -1.0 * (psk_double)M_PI ) )
      k_n += 1;
```

```
    out[idx] = v_in[idx] +
               ( ( (psk_double)k_n - (psk_double)k_p ) * 2.0 * (psk_double)M_PI );
  }
  free( in_diff );
  in_diff = NULL;

  return( out );
}

//------------------------------------------------------------------------
 psk_complex* psk_ff_lms( psk_complex* v_in1,
                          psk_complex* v_in2,
                          psk_uint32   len,
                          psk_uint32   n_taps )
{
  psk_uint32 i, k;
  i = k = 0;
  psk_complex* shift_reg = calloc( n_taps, sizeof(psk_complex) );
  psk_double lambda, tmp, tau, mu;
  lambda = tmp = tau = mu = 0.0;
  psk_double slowDown = 1.4; // default: 1.0
  psk_complex mu_in1 = psk_cmpl_mean( v_in1, len );
  psk_complex* h_est = calloc( n_taps, sizeof(psk_complex) );
  psk_complex* h_est_out = calloc( n_taps, sizeof(psk_complex) );


  for( i = 0; i < len; i++ )
  {
    tmp = psk_abs( psk_sub( v_in1[i], mu_in1 ) );
    lambda += pow( tmp, 2 );
  }
  lambda *= ( 1.0 / ( (psk_double)len - 1.0 ) );
  tau = (psk_double)len / ( 9.0 / slowDown );
  mu = ( 1.0 - expl( -1.0 / tau ) ) / lambda;

  psk_complex update_val, err, q;
  psk_complex* conj_shift_reg;
  psk_complex* q_vec;
  // initialize h_est
  RE( h_est[0] ) = 0.1;

  for( i = 0; i < len; i++ )
  {
    if( h_est_out )
    {
      free(h_est_out);
      h_est_out = NULL;
    }

    for( k = n_taps - 1; k > 0; k-- )
      shift_reg[k] = shift_reg[k - 1];

    shift_reg[0] = v_in1[i];

    q_vec = psk_cmpl_vec_mult( h_est, shift_reg, n_taps );
    RE( q ) = 0.0;
    IM( q ) = 0.0;
    for( k = 0; k < n_taps; k++ )
      q = psk_add( q, q_vec[k] );

    if( q_vec )
    {
      free( q_vec );
      q_vec = NULL;
    }

    err = psk_sub( v_in2[i], q );
    RE( err ) = RE( err ) * mu;
    IM( err ) = IM( err ) * mu;

    conj_shift_reg = psk_cmpl_vec_conj( shift_reg, n_taps );
```

```
    for( k = 0; k < n_taps; k++ )
    {
      update_val = psk_cmpl_mult( conj_shift_reg[k], err );
      h_est[k] = psk_add( h_est[k], update_val );
    }
    if( conj_shift_reg )
    {
      free( conj_shift_reg );
      conj_shift_reg = NULL;
    }
    h_est_out = psk_cmpl_vec_conj( h_est, n_taps );
  }
  // free memory
  if( h_est )
    free( h_est );
  if( shift_reg )
    free( shift_reg );

  return( h_est_out );
}
//-----------------------------------------------------------------------------
psk_int32 psk_get_phase_noise( psk_complex* sig,
                               psk_uint32   sig_len,
                               psk_double*  phase_noise )
{
  psk_double dpn = 0.0; // differential phase noise value
  psk_double downsample_factor = round( (psk_double)FS_INT * ETU );

  psk_uint32 start = 20; // disregard transient at beginning
  psk_uint32 step  = downsample_factor;
  psk_uint32 stop  = (psk_double)IDX_UNMOD;
  psk_uint32 len = 0;
  psk_uint32* idx_vec = psk_idx_linspace( start, step, stop, &len );
  if( idx_vec == NULL )
    return( PSK_ERR_OUT_OF_MEM );

  if( sig_len < stop || sig_len < len )
  {
    if( idx_vec )
      free( idx_vec );
    return( PSK_ERR_INVALID_SAMPLE_VEC );
  }

  // get the right symbols
  psk_complex iq_symbols[len];
  psk_uint32 idx;
  for( idx = 0; idx < len; idx++ )
    iq_symbols[idx] = sig[ idx_vec[idx] ];

  // get angles of elements
  psk_double* phi = NULL;
  phi = psk_vec_angle( iq_symbols, len );
  if( phi == NULL )
  {
    if( idx_vec )
      free( idx_vec );
    return( PSK_ERR_OUT_OF_MEM );
  }

  // get error of angles
  psk_double* phi_err = psk_diff( phi, len );
  if( phi_err == NULL )
  {
    if( idx_vec )
      free( idx_vec );
    return( PSK_ERR_OUT_OF_MEM );
  }

  // get rms value of phase noise.
  for ( idx = 0; idx < ( len - 1 ); idx++ )
```

```
    dpn += pow( phi_err[idx], 2.0 );

  dpn = sqrt( dpn / (psk_double)(len - 1) ) - 0.01 * ( (psk_double)FC/4.0 * (psk_double)
ETU) * (M_PI/180.0); // subtract machine-internal noise

  dpn /= ( EPI * (psk_double)M_PI / 180.0 );

  *phase_noise =  dpn ;

  // clean up
  if( phi )
    free( phi );

  if( phi_err )
    free( phi_err );

  if( idx_vec )
    free( idx_vec );

  return ( PSK_ERR_OK );
}

//---------------------------------------------------------------------------
psk_int32 psk_isi_param( psk_complex*  sig_bb_int,
                         psk_uint32    sig_bb_len,
                         psk_double*   isi_m,
                         psk_double*   isi_d,
                         psk_double*   phase_range,
                         psk_complex** sig_out )
{
  psk_uint32 idx;

  // used in loop to downsample the input signal
  psk_uint32 n_taps_k  = 4; // number of symbols
  psk_uint32 start_idx = (psk_double)FS_INT / ( 16.0 * (psk_double)FC );
  psk_uint32 end_idx   = (psk_double)sig_bb_len
                         - 16.0 * (psk_double)start_idx * (psk_double)FC * ETU;

  psk_uint32 nyquist_len = 16;
  psk_uint32 sps_p       = round( 16.0 * (psk_double)FC * ETU );
  psk_uint32* idx_vec    = NULL;

  const psk_int32* soc_lut; // constant array depending on ORDER
  if( ORDER == 8 )
    soc_lut = SOC_PSK8_deg;
  else if( ORDER == 16)
    soc_lut = SOC_PSK16_deg;
  else
    return(PSK_ERR_PARAMETER);

  psk_complex* estimate_symbol_resp = calloc( sps_p * n_taps_k,
                                              sizeof(psk_complex) );

  psk_complex* ref_sig = calloc( PSK_len, sizeof( psk_complex ) );
  for( idx = 0; idx < PSK_len; idx++ )
  {
    RE(ref_sig[idx]) = cos( (psk_double)soc_lut[idx]*(psk_double)M_PI / 180.0 );
    IM(ref_sig[idx]) = sin( (psk_double)soc_lut[idx]*(psk_double)M_PI / 180.0 );
  }

  // estimate symbol response
  for( idx = 0; idx < sps_p; idx++ )
  {
    psk_uint32 len = 0;
    idx_vec = psk_idx_linspace( start_idx + idx * start_idx,
                                start_idx * sps_p,
                                end_idx + ( idx + 1 ) * start_idx,
                                &len );
    psk_complex bb_out_grid_align_k[len];
    psk_uint32 cpy_idx;
    for( cpy_idx = 0; cpy_idx < len; cpy_idx++ )
```

```
    bb_out_grid_align_k[cpy_idx] = sig_bb_int[idx_vec[cpy_idx] - 1];


  psk_complex* hVec_k = psk_ff_lms( ref_sig,
                                    bb_out_grid_align_k,
                                    len,
                                    n_taps_k );
  // hVec_k = [a1 b1 c1 d1 ] if idx = 0
  // hVec_k = [a2 b2 c2 d2 ] if idx = 1 and so on
  // estimate_symbol_resp=[a1 a2 a3...][b1 b2 b3...][c1 c2 c3...][d1 d2 d3...]
  // because n_taps_k == 4 :
  estimate_symbol_resp[idx + (n_taps_k - 1) * sps_p] = hVec_k[(n_taps_k - 1)];
  estimate_symbol_resp[idx + (n_taps_k - 2) * sps_p] = hVec_k[(n_taps_k - 2)];
  estimate_symbol_resp[idx + (n_taps_k - 3) * sps_p] = hVec_k[(n_taps_k - 3)];
  estimate_symbol_resp[idx + (n_taps_k - 4) * sps_p] = hVec_k[(n_taps_k - 4)];

  if ( hVec_k )
    free( hVec_k );
}

psk_complex h[n_taps_k];
psk_complex tmp_sum;
RE( tmp_sum ) = 0.0;
IM( tmp_sum ) = 0.0;
psk_complex s0, s1;
RE( s0 ) = cosl( PR / 2.0 / 180.0 * M_PI );
IM( s0 ) = sinl( PR / 2.0 / 180.0 * M_PI );
s1 = psk_cmpl_conj( s0 );
psk_complex s_in[nyquist_len];

psk_double pr_comp; // stores current phase range
psk_complex s_in_pr[16];

for( idx = 0; idx < nyquist_len; idx++ )
  s_in[idx] = ( idx % 2 ) ? s0 : s1;

// s_in_pr = [s1 s1 s1 s1 s0 s0 s0 s0 s1 s1 s1 s1 s0 s0 s0 s0]
for( idx = 0; idx < 4; idx++ )
{
  s_in_pr[idx]     = s1;
  s_in_pr[idx + 8] = s1;
  s_in_pr[idx + 4] = s0;
  s_in_pr[idx + 12] = s0;
}

psk_complex* s_out = NULL;
psk_complex* pr_out = NULL;
psk_double L = 0.0;
psk_double isi_angle_1 = 0.0;
psk_double isi_angle_2 = 0.0;
psk_double isi_angle = 0.0;
psk_uint32 idx_max = 0;
psk_double* isi_m_vec = calloc( nyquist_len+2, sizeof(psk_double) );
psk_double* isi_d_vec = calloc( nyquist_len+2, sizeof(psk_double) );
psk_double* phase_range_vec = calloc( nyquist_len+2, sizeof(psk_double) );
psk_uint32 idx2=0;

for( idx = sps_p; idx > (sps_p - nyquist_len - 1) ; idx-- )
{
  RE( tmp_sum ) = 0.0;
  IM( tmp_sum ) = 0.0;

  // estimate_symbol_resp./sum(estimate_symbol_resp)
  tmp_sum = psk_add( tmp_sum, estimate_symbol_resp[ idx + 0 * sps_p - 1] );
  tmp_sum = psk_add( tmp_sum, estimate_symbol_resp[ idx + 1 * sps_p - 1] );
  tmp_sum = psk_add( tmp_sum, estimate_symbol_resp[ idx + 2 * sps_p - 1] );
  tmp_sum = psk_add( tmp_sum, estimate_symbol_resp[ idx + 3 * sps_p - 1] );
  h[0] = psk_cmpl_div( estimate_symbol_resp[ idx + 0 * sps_p - 1], tmp_sum );
  h[1] = psk_cmpl_div( estimate_symbol_resp[ idx + 1 * sps_p - 1], tmp_sum );
  h[2] = psk_cmpl_div( estimate_symbol_resp[ idx + 2 * sps_p - 1], tmp_sum );
  h[3] = psk_cmpl_div( estimate_symbol_resp[ idx + 3 * sps_p - 1], tmp_sum );
```

```
    // fir filter(h, s_in)
    s_out = psk_FIR_filter_cmpl( h, n_taps_k, s_in, nyquist_len );
    if( !s_out )
      return( PSK_ERR_OUT_OF_MEM );

    // phase range computation
    pr_out = psk_FIR_filter_cmpl( h, n_taps_k, s_in_pr, 16 );
    if ( !pr_out )
      return( PSK_ERR_OUT_OF_MEM );

    // compute phase range
    psk_complex pr_mean_high;
    RE( pr_mean_high ) = ( RE( pr_out[6] ) + RE( pr_out[7] ) + RE( pr_out[14] ) + RE( pr_
out[15] ) ) / 4.0;
    IM( pr_mean_high ) = ( IM( pr_out[6] ) + IM( pr_out[7] ) + IM( pr_out[14] ) + IM( pr_
out[15] ) ) / 4.0;
    psk_double pr_angle1 = atan2l( IM( pr_mean_high ), RE( pr_mean_high ) );

    psk_complex pr_mean_low;
    RE( pr_mean_low ) = ( RE( pr_out[2] ) + RE( pr_out[3] ) + RE( pr_out[10] ) + RE( pr_
out[11] ) ) / 4.0;
    IM( pr_mean_low ) = ( IM( pr_out[2] ) + IM( pr_out[3] ) + IM( pr_out[10] ) + IM( pr_
out[11] ) ) / 4.0;
    psk_double pr_angle2 = atan2l( IM( pr_mean_low ), RE( pr_mean_low ) );

    pr_comp = ( pr_angle1 - pr_angle2 ) * 180.0 / (psk_double)M_PI;


    psk_complex diff;
    psk_double tmp, max_val1, max_val2;
    tmp = max_val1 = max_val2 = 0.0;
    for( idx2 = 0; idx2 < nyquist_len; idx2++ )
    {
      // calculate isi_angle_1
      diff = psk_sub( s_out[idx2], s_in[idx2] );
      tmp = psk_abs( diff );
      if( tmp > max_val1 )
        max_val1 = tmp;

      //calculate isi_angle_2
      tmp  = atan2( IM( s_out[idx2] ), RE( s_out[idx2] ) );
      tmp -= atan2( IM( s_in[idx2] ),  RE( s_in[idx2] ) );
      tmp = ABS( tmp );
      if( tmp > max_val2 )
        max_val2 = tmp;
    }
    isi_angle_2 = max_val2;

    L = ( max_val1 > 0.99999 ) ? 0.99999 : max_val1;
    isi_angle_1 = asin( L );
    isi_angle = ( isi_angle_1 > isi_angle_2 ) ? isi_angle_1 : isi_angle_2;

    // find max isi_m value
    isi_m_vec[ sps_p - idx ] = isi_angle * 180.0 / M_PI / PR * ( ORDER - 1);
    isi_d_vec[ sps_p -idx ] = -( atan2( IM( h[1] ), RE( h[1] ) ) -
                              atan2( IM( h[0] ), RE( h[0] ) ) ) * 180.0 / M_PI;
    phase_range_vec[ sps_p - idx ] = pr_comp;

    if( s_out )
      free( s_out );
    s_out = NULL;

    if( pr_out )
      free( pr_out );
    pr_out = NULL;
  }

  for( idx = 0; idx < 3; idx++)
  {
    idx_max = 0;
```

```
    for( idx2 = 1; idx2 < nyquist_len+1; idx2 ++)
    {
      if( isi_m_vec[idx2] > isi_m_vec[idx_max] )
      {
        idx_max = idx2;

      }
    }
    *isi_m = isi_m_vec[idx_max];
    isi_m_vec[idx_max] = -1e302;
  }
  *isi_d = isi_d_vec[idx_max];
  *phase_range = phase_range_vec[idx_max];

  // isi_m, isi_d computation done. downsample sig_in
  psk_uint32 ds_start = (start_idx + sps_p - nyquist_len + idx_max) * start_idx;
  psk_uint32 ds_end   = (sig_bb_len / (start_idx * sps_p) ) * start_idx * sps_p;
  psk_uint32 ds_out_len = 0;
  psk_uint32* out_idx_vec = psk_idx_linspace( ds_start - 1,
                                              start_idx * sps_p,
                                              ds_end,
                                              &ds_out_len );
  psk_complex* sig_out_ = calloc( ds_out_len, sizeof(psk_complex) );
  for( idx = 0; idx < ds_out_len; idx++ )
    sig_out_[idx] =  sig_bb_int[out_idx_vec[idx] - 1] ;

  *sig_out = sig_out_;

  if( idx_vec )
    free( idx_vec );
  idx_vec = NULL;

  if( out_idx_vec )
    free( out_idx_vec );
  out_idx_vec = NULL;

  if( estimate_symbol_resp )
    free( estimate_symbol_resp );
  estimate_symbol_resp = NULL;

  return( PSK_ERR_OK );
}

//-----------------------------------------------------------------------
psk_int32 psk_re_align_symbol_grid( psk_complex* v_in,
                                    psk_uint32 len,
                                    psk_uint32* strt_idx,
                                    psk_uint32* end_idx )
{
  psk_uint32 idx, end_idx_1, strt_idx_1, sig_len, N_symb, sps;
  idx = end_idx_1 = strt_idx_1 = sig_len = N_symb = sps = 0;
  sps = round( (psk_double)FS_INT * ETU );
  psk_uint32 L_start = 50; // 50 symbols to remove beginning

  //-----------------------------------------------------------------------
  // (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
  //-----------------------------------------------------------------------

  // find approx start and end points of phase modulated signal
  psk_double offset = (24.0 - 8.0 ) * (psk_double)M_PI / 180.0;
  psk_double* phi = psk_vec_angle( v_in, len);


  //-----------------------------------------
  // (1) find and remove silence at the beginning, keep N etu's silence
  // before first modulation
```

```
  //-------------------------------------------

  // find beginning
  while( phi[idx] <= offset )
    idx++;

  strt_idx_1 = idx;

  // find end
  idx = 0;
  while( phi[ len - (1 + idx) ] <= offset )
    idx++;

  end_idx_1 = idx;


  if( strt_idx_1 > 25 * sps )
    strt_idx_1 = strt_idx_1 - 25 * sps + 1;
  else
    strt_idx_1 =0;

  if( end_idx_1 > 10 * sps )
    end_idx_1 = len - end_idx_1 - 2 + 10 * sps;
  else
    end_idx_1 = len - 1;

  sig_len = end_idx_1 - strt_idx_1 + 1;

  N_symb = (psk_double)sig_len / (psk_double)sps;
  if( len < PSK_len * sps )
    return( PSK_ERR_SIGNAL_TOO_SHORT );


  // normalize input signal
  psk_complex* v_in_norm = psk_cmpl_normalize( v_in, len );

  //-------------------------------------------
  // (2) take a initial random sampling phase
  //-------------------------------------------

  psk_uint32 t_idx_len = 0;
  psk_uint32* t_idx_vec = psk_idx_linspace( (strt_idx_1+sps) , sps, end_idx_1, &t_idx_
len);

  psk_complex* symb_rx = calloc( t_idx_len, sizeof(psk_complex) );
  for( idx = 0; idx < t_idx_len; idx++ )
    symb_rx[idx] = v_in_norm[ t_idx_vec[idx] ];



  // compute complex signal of SOC
  psk_complex* y_ref = calloc( PSK_len, sizeof(psk_complex) );
  psk_complex* y_ref_cut = calloc( PSK_len - L_start, sizeof(psk_complex) );
  const psk_int32* soc_lut;
  if( ORDER == 8 )
    soc_lut = SOC_PSK8_deg;
  else if( ORDER == 16)
    soc_lut = SOC_PSK16_deg;
  else
    return(PSK_ERR_PARAMETER);


  for( idx = 0; idx < PSK_len; idx++ )
  {
    RE(y_ref[idx]) = cosl( (psk_double)soc_lut[idx]*(psk_double)M_PI / 180.0 );
    IM(y_ref[idx]) = sinl( (psk_double)soc_lut[idx]*(psk_double)M_PI / 180.0 );
    if( idx > L_start - 1 )
    {
      RE( y_ref_cut[idx - L_start] ) = RE( y_ref[idx] );
      IM( y_ref_cut[idx - L_start] ) = IM( y_ref[idx] );
    }
```

```
  }


  psk_uint32 max_shift = N_symb - (psk_uint32)PSK_len+L_start;

  //-------------------------------------------
  // (3) Align on a symbol level
  //-------------------------------------------
  psk_uint32 delay_1 = 0;

  psk_int32 err = psk_find_approx_delay( y_ref_cut,
                                         PSK_len-L_start,
                                         symb_rx,
                                         t_idx_len,
                                         max_shift,
                                         &delay_1);

  if( err < 0)
    return(err);


  delay_1 -= (L_start-1); // corr. by L_start
  psk_uint32 idxBegin = strt_idx_1 + delay_1*sps - 1;
  psk_uint32 idxEnd   = idxBegin + (psk_uint32)PSK_len * sps - 1;


  if( y_ref_cut )
  {
    free( y_ref_cut );
    y_ref_cut = NULL;
  }

  if( symb_rx )
  {
    free( symb_rx );
    symb_rx = NULL;
  }

  //-------------------------------------------------------------------------
  // (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
  //-------------------------------------------------------------------

  psk_uint32 MAX_MISALIGN        = 2; //upper bound to input misalignment
  psk_uint32 FLAT_REGION_MIN     = 1; //algo specific, MIN 1, 2 safer; get at
  // least a full etu leading the beginning of symbol response
  psk_uint32 GuardTimeInterval   = 0; // 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.

  psk_uint32 Ns_m = idxEnd-idxBegin+1; // signal length
  psk_uint32 gridVals = ( 2 * MAX_MISALIGN + FLAT_REGION_MIN + 1 ) * sps; // idx length
  psk_uint32 N_samples = ( Ns_m - gridVals ) / sps; // number of samples used for
cross-covariance compuation
  psk_complex* futureXC = calloc( gridVals, sizeof(psk_complex) );
  psk_double* DelaySignature = calloc( gridVals, sizeof(psk_double) );

  // ------------------------------------------
  // (1) Get a pseudo symbol-response (signature1)
  // ------------------------------------------

  // start/stop index definition for y_ref signal
  // future  tx symbols
  psk_uint32 strt_idx_y_ref = FLAT_REGION_MIN + MAX_MISALIGN;
  psk_uint32 end_idx_y_ref = N_samples + 1;

  // prepare reference signal for cross-correlation operation
  psk_uint32 N_y_ref_cut = (end_idx_y_ref) - (strt_idx_y_ref + L_start) + 1;
  y_ref_cut = calloc( N_y_ref_cut, sizeof( psk_complex ) );
  psk_complex* y_ref_cut_conj = calloc( N_y_ref_cut, sizeof( psk_complex ) );
```

```
for( idx = strt_idx_y_ref + L_start-1; idx< end_idx_y_ref; idx++ )
{
  RE( y_ref_cut[ idx - (strt_idx_y_ref + L_start-1) ] ) = RE( y_ref[idx] );
  IM( y_ref_cut[ idx - (strt_idx_y_ref + L_start-1) ] ) = IM( y_ref[idx] );
}

psk_complex mu_x2 = psk_cmpl_mean( y_ref_cut, N_y_ref_cut );
for( idx = 0; idx < N_y_ref_cut; idx++ )
{
  y_ref_cut[ idx ] = psk_sub( y_ref_cut[ idx ], mu_x2 );
  y_ref_cut_conj[ idx ] = psk_cmpl_conj( y_ref_cut[ idx ] );
}

// length index for signal at fs
psk_uint32 const_end_offset_m = ( N_samples - MAX_MISALIGN - FLAT_REGION_MIN ) * sps;
psk_uint32 Lstart_m = L_start * sps; // length of periodic part in SOC at fs

psk_uint32 start, stop,iGrid;
start = stop = 0;
psk_uint32 len_vec_idx = 0;
psk_complex prod_tmp;
psk_uint32 *idx_vec = NULL;

for( iGrid = 0; iGrid < gridVals; iGrid++ )
{
  // compute start/stop indices
  start = idxBegin + iGrid - 1 + Lstart_m - sps;
  stop =  idxBegin + iGrid - 1 + const_end_offset_m;
  // get index vector
  idx_vec = psk_idx_linspace(start, sps, stop, &len_vec_idx);

  // cut out signal acc. to index vector
  symb_rx = calloc( len_vec_idx, sizeof(psk_complex) );
  for( idx = 0; idx < len_vec_idx; idx++ )
    symb_rx[idx] = v_in_norm[ idx_vec[idx] ];

  //compute and remove mean
  psk_complex mu_x1 = psk_cmpl_mean( symb_rx, len_vec_idx );
  for( idx = 0; idx < len_vec_idx; idx++ )
  {
    symb_rx[idx] = psk_sub( symb_rx[idx], mu_x1 );
    prod_tmp = psk_cmpl_mult( symb_rx[idx],  y_ref_cut_conj[idx] );

    futureXC[iGrid] = psk_add( futureXC[iGrid], prod_tmp );
  }
  DelaySignature[iGrid] = RE( futureXC[iGrid] );
  if( symb_rx )
  {
    free( symb_rx );
    symb_rx = NULL;
  }
  if( idx_vec )
  {
    free( idx_vec );
    idx_vec = NULL;
  }
}

// perform quality checks
psk_complex *ref_corr_0;
psk_int32 *ref_lag_0;
psk_double max_corr_val, peak_ref_0 ;
psk_uint32 max_corr_idx;

psk_int32 err1 = psk_cross_covariance( y_ref,
                                       PSK_len,
                                       y_ref,
                                       PSK_len,
                                       0,
                                       &ref_corr_0,
```

```
                                        &ref_lag_0 );

  if( err1 != PSK_ERR_OK )
    return( err1 );

  peak_ref_0 = psk_abs( ref_corr_0[0] );
  // ---------------------------------------------
  // (2) Find the peak in signature1
  // ---------------------------------------------

  psk_max( DelaySignature, gridVals, &max_corr_val, &max_corr_idx );

  if( max_corr_val < ( peak_ref_0 / 4.0 ) )
  {
    printf("\nSymbol Grid Alignment Failed, no reliable xcorr peak found.\n");
    return( PSK_SYMBOL_GRID_ALIGNMENT_FAIL );
  }

  if( futureXC )
    free( futureXC );

  //-----------------------------------------------------------------------
  // (3) Consider signature2 the slope (derivative) of signature1.
  //-----------------------------------------------------------------------

  psk_uint32 num_coef = 6;
  psk_double slope_coef[] = {1.0 / 12.0,
                             3.0 / 12.0,
                             2.0 / 12.0,
                            -2.0 / 12.0,
                            -3.0 / 12.0,
                            -1.0 / 12.0}; // normalized by sum(abs(nominator))

  psk_uint32 strt_idx_diff = num_coef +1;



  psk_double *DelaySignatureDiff = psk_FIR_filter( slope_coef,
                                                   num_coef,
                                                   DelaySignature,
                                                   gridVals );

  // ---------------------------------------------
  // (4) identify flat region in signature 2, to the left of the peak in signature1
  // flat region first guess based on signature 1 peak
  // ---------------------------------------------

  //point P1, earliest corner position
  psk_uint32 flatRegionCornerPosMin = max_corr_idx - (psk_uint32)( 1.3 * (psk_double)sps
);
  psk_uint32 idx1 = MAX( num_coef, (flatRegionCornerPosMin - sps) );

  // update start index:
  psk_uint32 end_flatregion = flatRegionCornerPosMin;
  psk_uint32 strt_idx_diff_idx = strt_idx_diff + idx1 - 1;

  // 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)
  psk_uint32 strt_idx_diff_emc = strt_idx_diff_idx + sps / 2;
  psk_uint32 end_flatregion_emc = end_flatregion + sps / 2;

  // cut out signal acc. to index vector
  len_vec_idx = end_flatregion - strt_idx_diff_idx + 1;
  psk_double *tmp_vec = calloc( len_vec_idx, sizeof(psk_double) );
  for( idx = strt_idx_diff_idx; idx <= end_flatregion; idx++ )
      tmp_vec[ idx-strt_idx_diff_idx ] = DelaySignatureDiff[ idx ];

  psk_double std_delSigDiff = psk_std(tmp_vec, len_vec_idx);
  std_delSigDiff *= 2.0;
```

```
if(tmp_vec)
{
  free(tmp_vec);
  tmp_vec = NULL;
}

// cut out signal acc. to index vector EMC based part
len_vec_idx = end_flatregion_emc - strt_idx_diff_emc+1;
tmp_vec = calloc( len_vec_idx, sizeof(psk_double) );
for( idx = strt_idx_diff_emc; idx <= end_flatregion_emc; idx++)
    tmp_vec[ idx - strt_idx_diff_emc ] = DelaySignatureDiff[ idx ];

psk_double std_delSigDiff_emc = psk_std(tmp_vec, len_vec_idx);

if ( std_delSigDiff_emc < std_delSigDiff )
{
  printf("\nOvershooting Channel suspected, improving flat region location\n");

  flatRegionCornerPosMin  = flatRegionCornerPosMin + sps / 4;
  idx1                    = MAX( num_coef, ( flatRegionCornerPosMin - sps ) );
  strt_idx_diff_idx = strt_idx_diff_emc;
  end_flatregion = end_flatregion_emc;

}

if( tmp_vec )
{
  free( tmp_vec );
  tmp_vec = NULL;
}

// ------------------------------------------------
// (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
// ------------------------------------------------

// cut out signal acc. to index vector EMC based part
len_vec_idx = end_flatregion - strt_idx_diff + 1;
tmp_vec = calloc( len_vec_idx, sizeof(psk_double) );
for( idx = strt_idx_diff; idx < end_flatregion; idx++)
    tmp_vec[ idx - strt_idx_diff ] = DelaySignatureDiff[ idx ];

psk_double MeanVal = psk_mean( tmp_vec, len_vec_idx );
psk_double MaxDev = psk_std(tmp_vec, len_vec_idx);
MaxDev *= 6.0;

if( tmp_vec )
{
  free( tmp_vec );
  tmp_vec = NULL;
}

//-----------------------------------------------------------
//(6) Reference Timing is @ the end of this flat region, corrected
//    for the derivative delay and a guard interval.
//-----------------------------------------------------------

idx = strt_idx_diff + flatRegionCornerPosMin;
psk_uint32 FLG = 0;
while( DelaySignatureDiff[ idx ] <= ( MeanVal + MaxDev ) )
{
  idx++;
  if( idx == gridVals )
  {
    break;
    FLG = 1;
  }
}

psk_uint32 idxPosFirstGuess = 0;
```

```
  if( FLG == 1 )
  {
    idxPosFirstGuess = flatRegionCornerPosMin;
  }
  else
  {
    idxPosFirstGuess = idx - strt_idx_diff + 1;
  }

  psk_uint32 backOff = GuardTimeInterval + 2; // (2..4) (==> slopeDelay: factor
                                              // two due to filter delay
                                              // max(1, floor((num_coef-1)/2)))
  psk_uint32 idxSymbolBegin = idxPosFirstGuess - backOff + strt_idx_diff - 1;

  // extract (re-Index) the sample aligned within input signal
  idxBegin -= (FLAT_REGION_MIN + MAX_MISALIGN) * sps;
  idxBegin += ( idxSymbolBegin );
  *strt_idx = idxBegin;
  *end_idx  = idxBegin + (psk_uint32)PSK_len * sps - 1;

  // clean up memory
  if( DelaySignature )
  {
    free( DelaySignature );
    DelaySignature = NULL;
  }

  if( DelaySignatureDiff )
  {
    free( DelaySignatureDiff );
    DelaySignatureDiff = NULL;
  }
  if( y_ref_cut )
  {
    free( y_ref_cut );
    y_ref_cut = NULL;
  }
  if( y_ref_cut_conj )
  {
    free( y_ref_cut_conj );
    y_ref_cut_conj = NULL;
  }
  return( PSK_ERR_OK );
}
//---------------------------------------------------------------------------
psk_int32 psk_find_approx_delay( psk_complex* v_ref_in,
                                 psk_uint32   ref_len,
                                 psk_complex* v_in,
                                 psk_uint32   len,
                                 psk_uint32   max_shift,
                                 psk_uint32*  delay )
{
  psk_uint32 delay_int, idx;
  psk_complex* v_corr;
  psk_int32* v_lag_idx;
  psk_double max_val = 0.0;
  psk_uint32 max_idx = 0;
  psk_int32 err = PSK_ERR_OK;
  psk_complex* v_in_aligned = calloc( ref_len, sizeof(psk_complex) );
  if( v_in_aligned == NULL )
    return( PSK_ERR_OUT_OF_MEM );

  if( len - ref_len < max_shift )
    printf("psk_find_approx_delay: v_in signal too short!");

  err = psk_cross_covariance( v_ref_in,
                              ref_len,
                              v_in,
                              len,
                              max_shift,
                              &v_corr,
```

```
                              &v_lag_idx );

  if (err != PSK_ERR_OK )
    return( err );

  psk_double* v_abs_corr = psk_vec_abs( v_corr, max_shift );

  psk_max( v_abs_corr, max_shift, &max_val, &max_idx );

  if( v_lag_idx[max_idx]> 1)
    printf("psk_find_approx_delay: akausal peak detected (peak > 1)");

  delay_int = -1 * v_lag_idx[max_idx];
  // signal alignment
  if( delay_int > 1 )
  {
    for( idx = delay_int; idx < delay_int+ref_len; idx++ )
    {
      RE( v_in_aligned[idx-delay_int] ) = RE( v_in[idx] );
      IM( v_in_aligned[idx-delay_int] ) = IM( v_in[idx] );
    }
  }

  // perform quality checks
  psk_complex *ref_corr_0;
  psk_int32 *ref_lag_0;
  psk_complex *corr_0;
  psk_int32 *lag_0;
  psk_double peak_ref_0, peak_0;

  err = psk_cross_covariance( v_ref_in,
                              ref_len,
                              v_ref_in,
                              ref_len,
                              0,
                              &ref_corr_0,
                              &ref_lag_0 );
  peak_ref_0 = psk_abs( ref_corr_0[0] );

  err = psk_cross_covariance( v_in_aligned,
                              ref_len,
                              v_in_aligned,
                              ref_len,
                              0,
                              &corr_0,
                              &lag_0 );
  peak_0 = psk_abs( corr_0[0] );

  if( max_val < 0.25 * sqrt( peak_ref_0 * peak_0) )
    printf("WARNING: psk_find_approx_delay -- x-corr peak low! \
            Signals do not match well.\n");

  // clean up
  if( v_abs_corr )
    free( v_abs_corr );
  if( v_in_aligned )
  free( v_in_aligned );
  if( v_corr )
    free( v_corr );
  if( v_lag_idx )
    free( v_lag_idx );

  *delay = delay_int;
  return( PSK_ERR_OK );
}

/*********************************************************************/
/*** psk_analysis.c                                                ***/
/***  DESCIRPTION:                                                 ***/
/***    Main file of the VHBR wave shape analysis tool             ***/
/*********************************************************************/
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

#include "psk_types.h"
#include "psk_defines.h"
#include "psk_math.h"
#include "psk_dsp.h"

/*****************************************************************/
/* function predeclaration */
/*****************************************************************/
/** readcsv
 * Read a ascii coded file of type csv. First column time, second column
 * amplitude.
 * @param in_filename Name of input file. Not longer than 256 characters
 * @param samples Array of amplitude values from input file
 * @param len Number of amplitude and time values
 * @param times Array of time values from input file
 * @return error code
 */
psk_int32 readcsv( char* in_filename,    /*[in]*/
                   psk_double** samples, /*[out]*/
                   psk_uint32* len,      /*[out]*/
                   psk_double** times);  /*[out]*/

psk_uint32 ORDER = 0;
psk_double BIT_RATE = 0.0;
psk_uint32 PR= 0;
psk_uint32 EPI = 0;
psk_double ETU = 0.0;

/*****************************************************************/
/** main
 * ISO/IEC 10373-6 VHBR PSK ANALYSIS TOOL
 * @param argc 2 parameters are expected
 * @param argv[1] expected to be a csv file
 * @param argv[2] expected to be a bit rate code [0=3fc/4, 1=fc, 2=3fc/2, 3=2fc]
 * @return error code as defined in psk_defines.h
 */
int main( int argc, char *argv[] )
{
  char in_filename[256];
  psk_uint32 samples_len = 0; // number of samples in input file
  psk_uint32 bitrate_code = -1;

  psk_double* samples = NULL;
  psk_double* times   = NULL;

  int status = PSK_ERR_OK;

  //
  // READ INPUT
  //
  printf("\n***********************************************");
  printf("\n****                                       ****");
  printf("\n**** ISO/IEC 10373-6 VHBR PSK ANALYSIS TOOL ****");
  printf("\n**** Version: 1.0 August 2013               ****");
  printf("\n****                                       ****");
  printf("\n***********************************************\n");

  if( argc == 1 )
  {
    printf("USAGE: \n");
    printf("1st parameter: file name, *.csv \n(1st column time vector, 2nd column
amplitude vector)\n\n");
    printf("2nd parameter: bit rate code [0=3fc/4, 1=fc, 2=3fc/2 or 3=2fc]\n\n");

    printf( "\nCSV File name :" );
```

```
    scanf( "%s", in_filename );

    printf( "\nBit rate code [0=3fc/4, 1=fc, 2=3fc/2 or 3=2fc] :" );
    scanf( "%d", &bitrate_code );
  }
  else
  {
    strcpy( in_filename, argv[1] );
    if( !strchr( in_filename, '.' ) )
      strcat( in_filename, ".csv" );

    if( argc > 2 )
    {
      char *br_code = argv[2];
      if( br_code[0] < '0' || br_code[0] > '3' )
      {
        printf( "\nERROR: valid bit rate codes are: [0=3fc/4, 1=fc, 2=3fc/2 or 3=2fc]\n"
);
        return( PSK_ERR_PARAMETER );
      }
      else
        bitrate_code = atoi( br_code );
    }
    else
    {
      printf( "\nBit rate code [0=3fc/4, 1=fc, 2=3fc/2 or 3=2fc] :" );
      scanf( "%d", &bitrate_code );
      if( bitrate_code < 0 || bitrate_code > 3 )
      {
        printf( "\nERROR: valid bit rate codes are: [0=3fc/4, 1=fc, 2=3fc/2 or 3=2fc]\n"
);
        return( PSK_ERR_PARAMETER );
      }
    }
  }

  if( !strchr( in_filename, '.' ) )
    strcat( in_filename, ".csv" );

  status = readcsv( in_filename,
                    &samples,
                    &samples_len,
                    &times);
  if( status != PSK_ERR_OK )
    return( status );


  if ( samples_len < MIN_NUM_SAMPLES )
    return( PSK_ERR_PARAMETER );

  // BIT RATE DEPENDENT PARAMETER
  switch( bitrate_code )
  {
  case 0:
    {
    ETU = 4.0 / (psk_double)FC;
    ORDER = 8;
    BIT_RATE = 3.0 * (psk_double)FC / 4.0;
    PR = 56; // degree
    EPI = 8; // degree
    break;
    }
  case 1:
    {
    ETU = 4.0 / (psk_double)FC;
    ORDER = 16;
    BIT_RATE = (psk_double)FC;
    PR = 60;
    EPI = 4;
    break;
    }
```

```
  case 2:
    {
    ETU = 2.0 / (psk_double)FC;
    ORDER = 8;
    BIT_RATE = 3.0 * (psk_double)FC / 2.0;
    PR = 56;
    EPI = 8;
    break;
    }
  case 3:
    {
    ETU = 2.0 / (psk_double)FC;
    ORDER = 16;
    BIT_RATE = 2.0*(psk_double)FC;
    PR = 60;
    EPI = 4;
    break;
    }
  default:
    {
      printf( "ERROR: No valid bit rate selected: %d\n", bitrate_code );
      printf( "\nBit rate code must be one of the following: [0=3fc/4, 1=fc, 2=3fc/2 or
3=2fc]. \n");
      return(PSK_ERR_PARAMETER);
    }
  }

  printf("Bit rate for evaluation is: %2.2f Mbps\n", BIT_RATE*1.0e-6 );

  //
  // PREPROCESSING AND CONDITIONING
  // STEP 1 - ANTIALIASING FILTER
  //
  psk_double* samples_aa = NULL;
  psk_uint32 aa_len = 0;
 // psk_double fs_osc = (psk_double)samples_len /
  //                     ( times[samples_len - 1] - times[0] );
  psk_double fs_osc = 1.0 /( times[1] - times[0] );

  if( FS_INT > fs_osc )
  {
    printf( "Target sampling rate must" );
    printf( " be equal to or smaller than initial sampling rate. " );
    printf( "Minimum initial sampling rate requirement: 500MSps!\n" );
    return( PSK_ERR_INVALID_SAMPLE_RATE );
  }
  status = psk_antialiasing( samples,
                             samples_len,
                             &samples_aa,
                             &aa_len,
                             fs_osc );
  if ( status != PSK_ERR_OK )
    return( status );

  //
  // PREPROCESSING AND CONDITIONING
  // STEP 2 - RESAMPLE
  //
  psk_uint32 len_int = 0; // length of resampled signal (integer number of fc)
  psk_double* times_int = NULL;
  psk_double* resampled_sig = NULL;
  status = psk_downsample( samples_aa,
                           times,
                           aa_len,
                           &resampled_sig,
                           &times_int,
                           &len_int );
  if ( status != PSK_ERR_OK )
    return( status );

  //
```

```
// PREPROCESSING AND CONDITIONING
// STEP 3 - DEMODULATE
//
psk_complex* sig_bb = NULL;
psk_demodulation( resampled_sig, times_int, len_int, &sig_bb );

// clean up unused variables and arrays
if( samples )
{
  free( samples );
  samples = NULL;
}
if( times )
{
  free( times );
  times = NULL;
}
if( samples_aa )
{
  free( samples_aa );
  samples_aa = NULL;
}
if( resampled_sig )
{
  free( resampled_sig );
  resampled_sig = NULL;
}

//
// PREPROCESSING AND CONDITIONING
// STEP 4 - DEROTATION
//
psk_complex* sig_bb_derot = NULL;
status = psk_derotation( sig_bb, times_int, len_int, &sig_bb_derot );

//
// PREPROCESSING AND CONDITIONING
// STEP 5 - LP-FILTER
//
psk_uint32 i;
psk_uint32 ma_filt_len = (FS_INT/FC)/2;
psk_uint32 len_out = 0;

psk_complex* sig_bb_lp = calloc( len_out, sizeof(psk_complex) );

status = psk_ma_filter_cmpl( sig_bb_derot,
                             len_int,
                             &sig_bb_lp,
                             &len_out,
                             ma_filt_len );
if( status != PSK_ERR_OK )
  return( status );

//
// PREPROCESSING AND CONDITIONING
// STEP 6 - SILENCE RE-ADJUSTMENT
//
psk_complex mean = psk_cmpl_mean( sig_bb_lp, IDX_UNMOD );
psk_double phi_err = atan2( IM( mean ), RE( mean ) );
psk_complex exp_phi_err;
RE( exp_phi_err ) = cos( phi_err );
IM( exp_phi_err ) = -sin( phi_err );

for( i = 0; i < len_out; i++ )
  sig_bb_lp[i] = psk_cmpl_mult( sig_bb_lp[i], exp_phi_err );

// clean up
if( sig_bb )
  free( sig_bb );
sig_bb = NULL;
```

```
  //
  // SYMBOL GRID ALIGNMENT
  // STEP 7 -
  //
  psk_uint32 strt_idx = 0;
  psk_uint32 end_idx = 0;

  status = psk_re_align_symbol_grid( sig_bb_lp, len_out,&strt_idx,&end_idx);

  if( status != PSK_ERR_OK)
    return( status );

  psk_uint32 sig_len_cutout = end_idx-strt_idx+1;
  psk_complex* sig_bb_aligned = calloc( sig_len_cutout, sizeof( psk_complex ) );

  for( i = strt_idx; i <= end_idx; i++ )
    sig_bb_aligned[i - strt_idx] = sig_bb_lp[ i ];


  //
  // ISI COMPUTATION
  // STEP 8 -
  //
  psk_double isi_m = 0.0;
  psk_double isi_d = 0.0;
  psk_double phase_range = 0.0;
  psk_complex* sig_out;
  status = psk_isi_param( sig_bb_aligned,
                         sig_len_cutout,
                          &isi_m,
                          &isi_d,
                          &phase_range,
                          &sig_out );
  if( status != PSK_ERR_OK )
    return( status );

  //
  // PHASE NOISE COMPUTATION
  // STEP 9 -
  //
  psk_double phase_noise = 0.0;
  status = psk_get_phase_noise( sig_bb_lp, strt_idx, &phase_noise);
  if( status != PSK_ERR_OK )
    return( status );


  printf("\n---------------------- STATISTICS --------------------------\n\n");
  printf("Input Sampling Rate: %d MHz\n", (psk_uint32)(fs_osc*1.0e-6 + 0.5) );
  printf("Number of Samples: %d \n\n", samples_len );

  printf("---------------------- RESULT --------------------------\n\n");
  printf("ISIm: %2.2E\n", isi_m);
  printf("ISId: %3.2f degrees\n", isi_d);
  printf("Phase range: %3.2f degrees\n", phase_range);
  printf("Normalized differential phase noise: %2.3E\n", phase_noise);

  return( status );
}

//-----------------------------------------------------------------------------
psk_int32 readcsv( char* in_filename,
                   psk_double** samples,
                   psk_uint32* len,
                   psk_double** times)
{
  psk_int32 num_samples = 1;
  psk_double* vt = calloc( num_samples, sizeof(psk_double) );
  psk_double* va = calloc( num_samples, sizeof(psk_double) );
  FILE *sample_file;

  // open input file
```

```
  if( !strchr( in_filename, '.' ) ) strcat( in_filename, ".csv" );
  if( ( sample_file = fopen( in_filename, "r" ) ) == NULL )
  {
    fprintf( stderr, "Cannot open input file %s.\n", in_filename );
    return( PSK_ERR_READ_FILE );
  }

  // read input values into array
  while( !feof( sample_file ) )
  {
    vt = (psk_double*)realloc( vt, sizeof(psk_double) * ( num_samples ) );
    va = (psk_double*)realloc( va, sizeof(psk_double) * ( num_samples ) );
    if( num_samples >= MAX_NUM_SAMPLES )
    {
      fprintf(stderr, "Too many samples in input file: only %d samples read\n",
        num_samples );
      break;
    }

    fscanf( sample_file, "%lf,%lf\n", &vt[num_samples-1], &va[num_samples-1] );
    num_samples++;
  }

  if( sample_file )
    fclose( sample_file );

  *len = num_samples - 1;
  *times = vt;
  *samples = va;

  return( PSK_ERR_OK );
}
```

K.3.13 Example test report

Below, an example test report for one test position in the operating volume and the bit rate of f_c 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 ISIm)

```
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
```

K.4 PCD signal creation for PICC reception tests

K.4.1 General

The following subclauses give an example of how to create test signals for PICC reception tests as required for conditions 1 to 4 of K.2.2.1.2. Test signals are digitally pre-conditioned before transmission.

K.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 $f_c/128$. 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:

a) the sequence of NPVs in its complex representation is filtered by H_{bb};

b) the carrier signal is phase modulated by the filtered NPVs.

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 ISIm and ISId 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 ISIm and ISId) is described in the z-domain by:

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

with p the complex pole.

The complex pole position p can be computed for desired ISIm and ISId parameters by:

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

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

K.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 second order, Butterworth type low-pass filter with 3 dB cut-off frequency of 100 kHz is used to filter frequency components above the cut-off frequency. The filter characteristic is illustrated in Figure K.7.

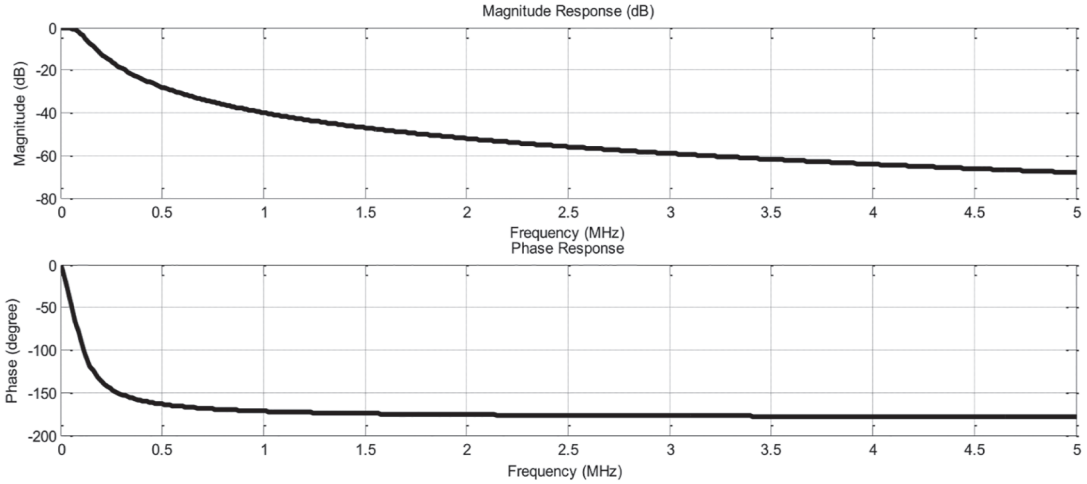


Figure K.7 — Filter characteristics of the second order Butterworth filter

K.4.4 PCD signal creation tool

This informative program written in ANSI C language describes how the four test conditions in the digital base-band domain could be digitally created for bit rates of $3f_c/4$, f_c, $3f_c/2$ and $2f_c$ from PCD to PICC. The implementation consists of a single file. Select one of the four test conditions and one of the four bit rates.

```
/*****************************************************************************/
/*** main.c                                                                ***/
/***  DESCRIPTION:                                                         ***/
/***  PCD-PICC main signal generation function for bit rates of            ***/
/***  3fc/4, fc, 3fc/2 and 2fc                                             ***/
/***  USAGE for: PICC reception tests                                      ***/
/***                                                                       ***/
/*****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <string.h>

#define PSK_ERR_OK                       0
#define PSK_ERR_PARAMETER               -1
#define PSK_ERR_OUT_OF_MEM              -2
#define PSK_ERR_LEN_MISMATCH            -3

#define RE(z) ( (z).re )
#define IM(z) ( (z).im )
#define BUTTER_SIZE_A 3
#define BUTTER_SIZE_B 3

// carrier frequency [Hz]
#define FC 13560000

typedef double psk_double;
typedef int psk_int32;
typedef unsigned int psk_uint32;

typedef struct
{
  psk_double re;
  psk_double im;
} psk_complex;

typedef struct
```

```
{
  psk_double a[BUTTER_SIZE_A];
  psk_double b[BUTTER_SIZE_B];
} psk_butter_coefs;

static const psk_int32  SOC_PSK8_deg[] = {0, 24, 24, -24, -24, 24, 24, -24, -24, 24, 24,
-24, -24, 24, 24, -24, -24, 24, 24, -24, -24, 24, 24, -24, -24, 24, 24, -24, -24, 24, 24,
-24, -24, 24, 24, -24, -24, 24, 24, -24, -24, 24, 24, -24, -24, 24, -24, 24, -24, 32, 32,
-24, 8, -16, 24, -8, 8, -16, 16, 16, 16, -24, 24, 32, 8, -8, 16, 8, -8, 16, 8, -16, 32,
-24, 16, 8, 8, -24, -16, 0, -8, -16, 24, -16, -8, 16, -16, 24, 8, 0, 32, 16, 32, -16, -16,
-24, 32, -8, -24, 8, -24, 8, 32, 8, -16, -16, 24, 24, 32, -16, 0, 32, -16, 8, -8, -16, 24,
24, 24, 16, -16, -24, 32, -24, 32, -24, 8, 24, 16, 0, 16, 24, -8, -24, 0, 32, 8, 8, 16, 8,
0};

static const psk_int32  SOC_PSK16_deg[] = {0, 28, 28, -28, -28, 28, 28, -28, -28, 28, 28,
-28, -28, 28, 28, -28, -28, 28, 28, -28, -28, 28, 28, -28, -28, 28, 28, -28, -28, 28, 28,
-28, -28, 28, 28, -28, -28, 28, 28, -28, -28, 28, 28, -28, -28, 28, -28, 28, -28, 32, 32,
-28, 8, -12, 32, 0, 16, -8, 28, 32, -28, -4, -20, -12, 28, 16, -20, -24, 24, -12, -20, 20,
4, 16, -8, -16, -16, 16, 28, -20, -28, 32, 8, -28, -16, 12, -16, 28, 16, 8, -20, 32, -12,
4, 4, -4, -12, 16, 4, -28, 8, -20, 4, -16, 28, 32, 8, 12, 20, -24, -4, 32, -16, 8, -8,
-12, 32, -28, -24, -28, 8, -28, 32, -28, 32, -28, 4, 24, 16, 0, 20, 32, 4, -12, 12, -20,
24, 28, -24, -28, 32};
static const psk_uint32 PSK_len = 141;


psk_uint32 ORDER    = 0;
psk_double BIT_RATE = 0.0;
psk_uint32 PR       = 0;
psk_uint32 EPI      = 0;
psk_double ETU      = 0.0;
psk_double FS       = 0.0;

psk_double D_PN_RMS = 0.035; // maximum normalized differential phase noise
                             // (PICC RX)[EPI]

/*****************************************************************************/
/**
 * psk_isi_pole
 *  computes ISI filter coefficients
 *  @param isi_m Inter-symbol interference magnitude [EPI]
 *  @param isi_d Inter-symbol interference rotation [degree]
 *  @param b Pointer to variable where to store the feed forward filter coefs
 *  @param a Pointer to array where to store the feedback filter coefs
 *  @return error value as defined at the top of this file
 */
psk_int32 psk_isi_pole( psk_double isi_m,  /*[in]*/
                        psk_double isi_d,  /*[in]*/
                        psk_complex* b,    /*[out]*/
                        psk_complex** a ); /*[out]*/

/*****************************************************************************/
/**
 * psk_awgn_phase_noise
 *  get LP-filtered AWG noise normalized to d_pn_rms
 *  @param sig_len Required vector length of noise
 *  @param d_pn_rms Target normalized differential phase noise [rms]
 *  @param out Pointer to array where to store the phase noise
 *  @return error value as defined at the top of this file
 */
psk_int32 psk_awgn_phase_noise( psk_uint32 sig_len,  /*[in]*/
                                psk_complex** out ); /*[out]*/

/*****************************************************************************/
/**
 * psk_mean
 *  calculate the arithmetic mean of a given vector
 *  @param vec Calculate the mean of the values in this vector
 *  @param len The vector's number of elements
 *  @return The aritmethic mean or zero, if len < 2
 */
psk_double psk_mean( psk_double* vec /*[in]*/, psk_uint32 len /*[in]*/);
```

```
/*****************************************************************************/
/**
 * psk_diff
 *  The resulting vector's elements are the differences of two consecutive
 *  elements of a given vector. The resulting vector has a length of len-1
 *  @param vec Calculate the consecutive differences of this vector's values
 *  @param len The vector's number of elements
 *  @return The aritmethic mean or zero, if len < 2
 */
psk_double* psk_diff( psk_double* vec /*[in]*/, psk_uint32 len /*[in]*/);

/*****************************************************************************/
/**
 * psk_compute_butter_lp_coef
 *  Computes the low-pass filter coefficients of a 4th order Butterworth
 *  IIR filter with a cut-off frequency of 120MHz
 *  @param fs Sampling rate
 *  @return Struct with the b and a polynom coefficients
 */
psk_butter_coefs psk_compute_butter_lp_coef( psk_double sample_rate /*[in]*/);

/*****************************************************************************/
/**
 * psk_antialiasing filter
 *  Filters the signal with a 2th order IIR filter
 *  @param v_in Input signal amplitude
 *  @param len_in Length of v_in
 *  @param v_out_ptr Pointer to output array
 *  @param len_out Length of output array
 *  @param sampling rate
 *  @return error value
 */
psk_int32 psk_antialiasing( psk_double* v_in        /*[in]*/,
                            psk_uint32 len_in       /*[in]*/,
                            psk_double **v_out_ptr  /*[out]*/,
                            psk_uint32* len_out     /*[out]*/);

/*****************************************************************************/
/**
 * psk_ISI_filter
 *  Filters the signal with ISI filter coefficients
 *  @param v_in Input signal amplitude in complex base bans
 *  @param len_in Length of v_in
 *  @param v_out_ptr Pointer to output array
 *  @param len_out Length of output array
 *  @param sampling rate
 *  @return error value
 */
psk_int32 psk_ISI_filter(   psk_complex* v_in         /*[in]*/,
                            psk_uint32 len_in         /*[in]*/,
                            psk_complex b             /*[in]*/,
                            psk_complex *a            /*[in]*/,
                            psk_complex **v_out_ptr   /*[out]*/,
                            psk_uint32* len_out       /*[out]*/);

/*****************************************************************************/
/**
 * psk_cmpl_mult
 *  Calculate the product of two complex numbers
 *  @param a First complex number
 *  @param b Second complex number
 *  @return The complex result
 */
psk_complex psk_cmpl_mult( psk_complex a /*[in]*/, psk_complex b /*[in]*/ );

/*****************************************************************************/
/**
 * psk_add
 *  Calculate the sum of two complex numbers
 *  @param a First summand
```

```
 *  @param b Second summand
 *  @return The complex result
 */
psk_complex psk_add( psk_complex a /*[in]*/, psk_complex b /*[in]*/ );

/*****************************************************************/
/**
 * psk_sub
 *  Calculate the difference of two complex numbers
 *  @param a Minuend
 *  @param b Subtrahend
 *  @return The complex result
 */
psk_complex psk_sub( psk_complex a /*[in]*/, psk_complex b /*[in]*/ );

/*****************************************************************/
/*****************************************************************/
/*****************************************************************/
/*****************************************************************/
/*****************************************************************/
int main( int argc, char *argv[] )
{
  psk_uint32 idx, out_len;
  psk_complex* out_sig = NULL;
  psk_int32 status = 0;
  psk_double ISIm = 0.0; // [EPI]
  psk_double ISId = 0.0;
  psk_complex b;
  psk_complex* a;
  psk_complex* noise = NULL;
  const psk_int32* soc_lut; // constant array depending on ORDER
  printf("\n*****************************************************");
  printf("\n****                                             ****");
  printf("\n**** ISO/IEC 10373-6 VHBR PSK SIGNAL CREATION TOOL ****");
  printf("\n**** Version: 1.0 AUGUST 2013                     ****");
  printf("\n****                                             ****");
  printf("\n*****************************************************\n");

  if ( argc != 3 )
  {
    printf("Usage: signal creation:\n");
    printf("       1st parameter: condition[1, 2, 3, 4]\n");
    printf("       2nd parameter: bitrate code [0=3fc/4, 1=fc, 2=3fc/2 or 3=2fc]\n");
    return ( PSK_ERR_PARAMETER );
  }

  psk_uint32 bitrate_code = atoi( argv[2] );

// BIT RATE DEPENDENT PARAMETER
  switch( bitrate_code )
  {
    case 0:
    {
      ETU = 4.0 / (psk_double)FC;
      ORDER = 8;
      BIT_RATE = 3.0 * (psk_double)FC / 4.0;
      PR = 56;
      EPI = 8;
      break;
    }
    case 1:
    {
      ETU = 4.0 / (psk_double)FC;
      ORDER = 16;
      BIT_RATE = (psk_double)FC;
      PR = 60;
      EPI = 4;
      break;
    }
    case 2:
    {
```

```
      ETU = 2.0 / (psk_double)FC;
      ORDER = 8;
      BIT_RATE = 3.0 * (psk_double)FC / 2.0;
      PR = 56;
      EPI = 8;
      break;
    }
    case 3:
    {
      ETU = 2.0 / (psk_double)FC;
      ORDER = 16;
      BIT_RATE = 2.0*(psk_double)FC;
      PR = 60;
      EPI = 4;
      break;
    }
    default:
    {
      printf( "ERROR: No valid bit rate selected [0=3fc/4, 1=fc, 2=3fc/2 or 3=2fc]: %c\n",
bitrate_code );
      return ( PSK_ERR_PARAMETER );
    }
  }
  FS = 1.0/ETU;

  printf("Bit rate for evaluation is: %2.2f Mbps\n", BIT_RATE*1.0e-6 );

  if( strcmp( argv[1], "condition1" ) == 0 )
  {
    ISIm = 1.1; // [EPI]
    ISId = 45; // [degree]
    status = psk_isi_pole( ISIm, ISId, &b, &a );

    if (status != PSK_ERR_OK)
      return(status);
  }
  else if ( strcmp( argv[1], "condition2" ) == 0 )
  {
    ISIm = 1.1; // [EPI]
    ISId = -45; // [degree]

    status = psk_isi_pole( ISIm, ISId, &b, &a );

    if (status != PSK_ERR_OK)
      return(status);
  }
  else if ( strcmp( argv[1], "condition3" ) == 0 )
  {
    ISIm = 0.6; // [EPI]
    ISId = 120; // [degree]
    status = psk_isi_pole( ISIm, ISId, &b, &a );

    if (status != PSK_ERR_OK)
      return(status);
  }
  else if ( strcmp( argv[1], "condition4" ) == 0 )
  {
    ISIm = 1.6; // [EPI]
    ISId = 0; // [degree]
    status = psk_isi_pole( ISIm, ISId, &b, &a );

    if (status != PSK_ERR_OK)
      return(status);
  }
  else
  {
    printf( "ERROR: No valid test condition selected!\n" );
    return( PSK_ERR_PARAMETER );
  }
```

```
  if( ORDER == 8 )
    soc_lut = SOC_PSK8_deg;
  else if( ORDER == 16)
    soc_lut = SOC_PSK16_deg;
  else
    return(PSK_ERR_PARAMETER);

  psk_uint32 idx_offset = 2;
  psk_complex* ref_sig = calloc( (PSK_len + idx_offset) , sizeof( psk_complex ) );
  for(idx = 0; idx < (PSK_len + idx_offset); idx++)
  {
    if (idx < idx_offset)
      RE(ref_sig[idx]) = 1.0;
    else
    {
      RE(ref_sig[idx]) = cos( (psk_double)soc_lut[ idx - idx_offset ] * (psk_double)M_PI /
180.0 );
      IM(ref_sig[idx]) = sin( (psk_double)soc_lut[ idx - idx_offset ] * (psk_double)M_PI /
180.0 );
    }

  }

  status = psk_ISI_filter( ref_sig, ( PSK_len + idx_offset ) , b, a, &out_sig, &out_len );

  if ( status != PSK_ERR_OK )
      return( status );

  if( out_len != PSK_len)
    return( PSK_ERR_LEN_MISMATCH );

  status = psk_awgn_phase_noise( PSK_len, &noise );

  if ( status != PSK_ERR_OK )
      return( status );


  for( idx = 0; idx < PSK_len; idx++ )
    out_sig[ idx ] = psk_cmpl_mult( out_sig[ idx ], noise[ idx ] );

  char out_filename[ 35 ];
  strcpy(out_filename, "sig_");
  strcat( out_filename , argv[1] );
  strcat( out_filename , "_bit_rate_code_" );
  strcat( out_filename , argv[2] );
  strcat( out_filename , ".txt" );

  printf("file name: %s\n", out_filename);
  // writecsv
  FILE* fid = fopen( out_filename, "w" );
  for( idx = 0; idx < out_len; idx++ )
    fprintf( fid, "%4.8E,%4.8E\n", RE( out_sig[idx] ), IM( out_sig[idx] ) );

  fclose(fid);

  if( ref_sig )
    free(ref_sig);

  if( out_sig )
    free(out_sig);

  if( noise )
    free(noise);

  return( PSK_ERR_OK );
}


//---------------------------------------------------------------------
psk_int32 psk_isi_pole( psk_double isi_m,
                        psk_double isi_d,
```

```
                        psk_complex* b,
                        psk_complex** a )
{
  psk_double fc = (psk_double)FC;
  psk_double Tsr = 1.0 / (psk_double)FS;
  psk_double Tc = 1.0 / fc;
  psk_double phase_range = (psk_double)PR * (psk_double)M_PI / 180.0;
  psk_double epi = (psk_double)EPI  * (psk_double)M_PI / 180.0;

  psk_double exponent = Tsr / ( ETU - Tc );
  psk_double basis = sinl( isi_m * epi ) / ( 2.0 * sinl( phase_range / 2.0 ) );
  psk_complex p;
  psk_complex b_;
  psk_complex *a_ = calloc( 2, sizeof( psk_complex ));

  RE( p ) = pow( basis, exponent ) * cosl( M_PI / 180.0 * isi_d * exponent );
  IM( p ) = pow( basis, exponent ) * sinl( M_PI / 180.0 * isi_d * exponent );

  RE( b_ ) = 1.0;
  IM( b_ ) = 0.0;
  RE(b_)  = RE(b_) - RE( p ) ;
  IM(b_)  = -1.0 * IM( p ) ;
  RE( a_[0] ) = 1.0;
  IM( a_[0] ) = 0.0;
  RE( a_[1] ) = ( -1.0 ) * RE( p );
  IM( a_[1] ) = ( -1.0 ) * IM( p );

  *a = a_;
  *b = b_;

  return ( PSK_ERR_OK );
}

//---------------------------------------------------------------------------
psk_int32 psk_awgn_phase_noise( psk_uint32 sig_len,
                                psk_complex** out )
{
  psk_double *noise = calloc(sig_len + BUTTER_SIZE_A, sizeof( psk_double) );
  psk_complex *out_ = calloc(sig_len+1, sizeof(psk_complex) );
  psk_int32 samp = 0;
  psk_uint32 idx, len_out;
  psk_double *n_out_vec = NULL;
  psk_double d_pn, fact;
  psk_double epi = (psk_double)EPI * M_PI/180.0;
  const psk_uint32 MAX_RAND_NUM = 100;

  srand(time(0));
  // get random numbers
  for(idx = 0; idx < (sig_len + BUTTER_SIZE_A); idx++)
  {
    samp = rand() % MAX_RAND_NUM;
    noise[idx] = 2.0 * (psk_double)samp / (psk_double)(MAX_RAND_NUM);
  }

  // estimate and remove mean
  psk_double mu_noise = psk_mean(noise, sig_len+BUTTER_SIZE_A);

  for(idx = 0; idx < (sig_len + BUTTER_SIZE_A); idx++)
    noise[idx] -= mu_noise;


  psk_int32 stat = psk_antialiasing( noise,
                                     sig_len+BUTTER_SIZE_A,
                                     &n_out_vec,
                                     &len_out );

  if( stat!=PSK_ERR_OK )
    return( stat );

  psk_double *n_diff_vec = psk_diff( n_out_vec, len_out );
```

```
  mu_noise = 0.0;
  for(idx = 0; idx < ( len_out-1 ) ; idx++ )
  {
    n_diff_vec[idx] = pow( n_diff_vec[idx], 2 );
    mu_noise += n_diff_vec[idx];
  }

  mu_noise /= ( len_out-1 );
  d_pn = sqrt( mu_noise ) / epi;

  fact = D_PN_RMS / d_pn;

  for(idx = 0; idx < len_out ; idx++)
  {
    n_out_vec[idx] *= fact;
    RE( out_[idx] ) = cosl( n_out_vec[idx] );
    IM( out_[idx] ) = sinl( n_out_vec[idx] );
  }

  if( noise )
    free(noise);

  if( n_out_vec )
    free(n_out_vec);

  if( n_diff_vec )
    free(n_diff_vec);

  *out = out_;
  return( PSK_ERR_OK );
}

//---------------------------------------------------------------------------
psk_double psk_mean( psk_double* vec, psk_uint32 len )
{
  psk_double sum = 0.0;
  psk_uint32 idx;

  if( len < 2 )
    return sum;

  for( idx = 0; idx < len; idx++ )
    sum += vec[idx];

  sum /= (double)len;

  return(sum);
}

//---------------------------------------------------------------------------
psk_double* psk_diff( psk_double* vec , psk_uint32 len )
{
  psk_double* diff = calloc( ( len - 1 ), sizeof(psk_double) );
  psk_uint32 idx;

  for( idx = 0; idx < ( len - 1 ); idx++ )
    diff[idx] = vec[idx + 1] - vec[idx];

  return(diff);
}

//---------------------------------------------------------------------------
psk_int32 psk_antialiasing( psk_double* v_in,
                            psk_uint32 len_in,
                            psk_double **v_out_ptr,
                            psk_uint32* len_out )
{

  psk_double len_butter_out = (psk_double)len_in - (psk_double)BUTTER_SIZE_A;
  psk_double *v_out = calloc( len_butter_out,  sizeof(psk_double) );
```

```
  if( v_out == NULL )
    return( PSK_ERR_OUT_OF_MEM );

  psk_double *out_tmp = calloc( sizeof(psk_double), len_in );
  if( out_tmp == NULL )
  {
    if( v_out )
      free( v_out );

    return( PSK_ERR_OUT_OF_MEM );
  }
  psk_int32 i = 0;
  psk_uint32 t_0 = 0;
  psk_uint32 t_1 = 0;
  psk_uint32 t_2 = 0;

  psk_butter_coefs coefs;


  coefs = psk_compute_butter_lp_coef( FS );
  for( i = 0; i < len_in; i++ )
  {
    if( i < BUTTER_SIZE_A-1)
    {
      t_0 = fmax( 0, i );
      t_1 = fmax( 0, i - 1 );
      t_2 = fmax( 0, i - 2 );

      out_tmp[i] = coefs.b[0] * v_in[t_0] + coefs.b[1] * v_in[t_1] +
                coefs.b[2] * v_in[t_2]     -
                ( coefs.a[0] * out_tmp[t_0] + coefs.a[1] * out_tmp[t_1] +
                coefs.a[2] * out_tmp[t_2] );
    }
    else
    {
      out_tmp[i] = coefs.b[0] * v_in[i] + coefs.b[1] * v_in[i - 1] +
                coefs.b[2] * v_in[i - 2]  -
                ( coefs.a[0] * out_tmp[i] + coefs.a[1] * out_tmp[i - 1] +
                coefs.a[2] * out_tmp[i - 2] );
    }
    if( i > BUTTER_SIZE_A - 1 )
    {
      v_out[i - BUTTER_SIZE_A] = out_tmp[i];
    }
  }

  *v_out_ptr = v_out;
  *len_out = len_butter_out;

  if( out_tmp )
    free( out_tmp );
  out_tmp = NULL;

  return( PSK_ERR_OK );
}

//-----------------------------------------------------------------------------
psk_int32 psk_ISI_filter(   psk_complex* v_in,
                            psk_uint32 len_in,
                            psk_complex b,
                            psk_complex *a,
                            psk_complex **v_out_ptr,
                            psk_uint32* len_out )
{
  psk_uint32 len_a = 2;
  psk_uint32 len_sig_out = len_in - len_a;
  psk_complex *v_out1 = calloc( len_sig_out+1, sizeof(psk_complex) );

  if( v_out1 == NULL )
    return( PSK_ERR_OUT_OF_MEM );
```

```
  psk_complex *out_tmp = calloc( len_in, sizeof(psk_complex) );
  if( out_tmp == NULL )
  {
    if( v_out1 )
      free( v_out1 );

    return( PSK_ERR_OUT_OF_MEM );
  }
  psk_int32 i = 0;
  psk_uint32 t_0 = 0;
  psk_uint32 t_1 = 0;

  psk_complex tmp_a0, tmp_a1, tmp_b0;

  for( i = 0; i < len_in; i++ )
  {
    if( i < len_a-1)
    {
      t_0 = fmax( 0, i );
      t_1 = fmax( 0, i - 1 );
      tmp_b0 = psk_cmpl_mult( b, v_in[t_0] );
      tmp_a0 = psk_cmpl_mult( a[0], out_tmp[t_0] );
      tmp_a1 = psk_cmpl_mult( a[1], out_tmp[t_1] );

      out_tmp[i] = psk_sub( tmp_b0, psk_add( tmp_a0, tmp_a1 ) );
    }
    else
    {
      tmp_b0 = psk_cmpl_mult( b, v_in[i] );
      tmp_a0 = psk_cmpl_mult( a[0], out_tmp[i] );
      tmp_a1 = psk_cmpl_mult( a[1], out_tmp[ i - 1 ] );

      out_tmp[i] = psk_sub( tmp_b0, psk_add( tmp_a0, tmp_a1 ) );
    }
    if( i > len_a - 1 )
    {
      v_out1[i - len_a] = out_tmp[i];
    }
  }

  *v_out_ptr = v_out1;
  *len_out = len_sig_out;

  if( out_tmp )
  {
    free( out_tmp );
    out_tmp = NULL;
  }


  return( PSK_ERR_OK );
}

//---------------------------------------------------------------------------
psk_butter_coefs psk_compute_butter_lp_coef( psk_double sample_rate )
{
  psk_uint32 i = 0;
  psk_uint32 butter_order = ( BUTTER_SIZE_A - 1 );
  psk_double wd = 100e3 * 2 * M_PI; // cutoff frequency in radians ==> 120 Mhz
  psk_double Ts = 1.0 / (psk_double)sample_rate;
  psk_double wa = 0.0;
  //psk_double wa1 = 0.0;
  psk_double wa_p2 = 0.0;
  psk_double g = 0.0;
  psk_double a_biquad[3*butter_order/2];
  psk_double b_biquad[3*butter_order/2];
  psk_double norm_fact = 0.0;

  psk_butter_coefs coefs;

  memset( a_biquad, 0, 3*butter_order/2 );
```

```
  memset( b_biquad, 0, 3*butter_order/2 );
//  memset( coefs, 0, sizeof(psk_butter_coefs) );

  if(butter_order == 0) // filter order must be greater then Zero and even
  {
    return(coefs);
  }

  if( butter_order % 2 ) // filter order must be even
  {
    return(coefs);
  }

  // Step 1: pre-warping of cut-off frequency
  wa = tan( wd * Ts / 2.0 );
  //wa1 = 2.0 / Ts * wa;

  wa_p2 = pow( wa, 2);
  // Step 2: compute biquad filter coefficients

  for( i = 0 ; i < ( butter_order / 2 ); i++ )
  {
    norm_fact = 1.0 + 2.0 * cos( M_PI * ( 2.0 * (psk_double)i + 1.0 ) /
                ( 2.0 * (psk_double)butter_order) ) * wa + wa_p2 ;
    g = wa_p2 / norm_fact;

    b_biquad[i*3]   = 1.0 * g;
    b_biquad[(i*3 + 1)] = 2.0 * g;
    b_biquad[(i*3 + 2)] = (1.0 *g);

    a_biquad[i*3]   = 1.0;
    a_biquad[i*3+1] = ( 2.0 * wa_p2 - 2.0 )/ norm_fact ;
    a_biquad[i*3+2] = ( 1.0 - 2.0 * cos(M_PI * ( 2.0 * (psk_double)i + 1.0 ) /
              ( 2.0 * (psk_double)butter_order ) ) * wa + wa_p2 ) / norm_fact;

  }


  // compute polynomial from 2 biquads (SOS)
  if( butter_order == 2 )
  {
    coefs.b[0] = b_biquad[0];
    coefs.b[1] = b_biquad[1];
    coefs.b[2] = b_biquad[2];

    coefs.a[0] = a_biquad[0];
    coefs.a[1] = a_biquad[1];
    coefs.a[2] = a_biquad[2];
  }
  else if( butter_order == 4 )
  {
    //(a0*s^2+a1*s+a2)*(b0*s^2+b1*s+b2)= s^4(a0*b0)+s^3(a1*b0+a0*b1)+s^2(a2*b0+a1*b1+a0*b2
)+s(a2*b1+a1*b2)+a2*b2
    coefs.b[0] = b_biquad[0] * b_biquad[3];
    coefs.b[1] = b_biquad[1] * b_biquad[3] + b_biquad[0] * b_biquad[4];
    coefs.b[2] = b_biquad[2] * b_biquad[3] + b_biquad[1] * b_biquad[4] + b_biquad[0] * b_
biquad[5];
    coefs.b[3] = b_biquad[2] * b_biquad[4] + b_biquad[1] * b_biquad[5];
    coefs.b[4] = b_biquad[2] * b_biquad[5];

    coefs.a[0] = a_biquad[0] * a_biquad[3];
    coefs.a[1] = a_biquad[1] * a_biquad[3] + a_biquad[0] * a_biquad[4];
    coefs.a[2] = a_biquad[2] * a_biquad[3] + a_biquad[1] * a_biquad[4] + a_biquad[0] * a_
biquad[5];
    coefs.a[3] = a_biquad[2] * a_biquad[4] + a_biquad[1] * a_biquad[5];
    coefs.a[4] = a_biquad[2] * a_biquad[5];

  }

  return(coefs);
}
```

```
//-----------------------------------------------------------------------
psk_complex psk_cmpl_mult( psk_complex a, psk_complex b )
{
  psk_complex prod;
  RE( prod ) = RE( a ) * RE( b ) - IM( a ) * IM( b );
  IM( prod ) = IM( a ) * RE( b ) + RE( a ) * IM( b );

  return(prod);
}

//-----------------------------------------------------------------------
psk_complex psk_add( psk_complex a, psk_complex b )
{
  psk_complex sum;
  RE( sum ) = RE( a ) + RE( b );
  IM( sum ) = IM( a ) + IM( b );

  return(sum);
}

//-----------------------------------------------------------------------
psk_complex psk_sub( psk_complex a, psk_complex b )
{
  psk_complex diff;
  RE( diff ) = RE( a ) - RE( b );
  IM( diff ) = IM( a ) - IM( b );

  return(diff);
}
```

Annex L
(normative)

Frame with error correction test methods

L.1 Frame format selection

L.1.1 PCD tests

L.1.1.1 Scope

This test is to verify the proper selection and usage of frames with error correction including framing options. These tests apply only to PCDs which support frames with error correction in at least one direction.

L.1.1.2 Apparatus

In this test the PCD-test-apparatus shall be configurable to change the frame format during the test procedure. The PCD-test-apparatus shall be able to monitor the used frames on each stage of this test procedure.

L.1.1.3 Procedure

The following procedure shall be repeated for all cases defined in Table L.1, Table L.2 and Table L.3. In each test case the PCD-test-apparatus indicates the support of all optional bit rates. Monitor the used frame format and framing options.

Cases 1 to 6 (see Table L.1) shall be applied for Type A and Type B. No framing options shall be indicated (by different methods).

Cases 7 to 11 (see Table L.2) shall be applied for Type A, for PCD to PICC bit rates greater than $f_c/16$. Frames with error correction shall be indicated as independently optional in each direction (b2=(1)b and b8=(0)b) and may be activated. If the PCD does not support PCD to PICC bit rates greater than $f_c/16$, these cases shall be omitted.

Cases 12 to 23 (see Table L.3) shall be applied for Type B. Frames with error correction shall be indicated as independently optional in each direction (b2=(1)b and b8=(0)b) and may be activated.

a) The UT performs the protocol activation procedure according to H.1.9.2 for Type A or H.1.9.3 for Type B. When the PCD is embedded in a product or the negotiation does not start immediately after protocol activation procedure, the method to activate the mechanism should be provided by the PCD manufacturer to adapt the test scenario accordingly.

b) The PCD shall send an S(PARAMETERS) block to request frame format parameters.

c) The LT answers with an S(PARAMETERS) block including frame format indication bytes according to Table L.1 and Table L.2 for Type A and Table L.1 and Table L.3 for Type B.

NOTE 1 Only a subset of all possible options is tested.

d) The PCD shall send an S(PARAMETERS) block with a INF field containing frame format activation bytes with

1) exactly one bit set for frame format from PCD to PICC,

2) and exactly one bit set for frame format from PICC to PCD,

3) and may set some bits from framing options indicated by the LT in step c).

e) The LT acknowledges the received S(PARAMETERS) block with an S(PARAMETERS) block response.

f) PCD shall send $I(0)_0$ block using the frame format selected. This block may also be $I(1)_0$, S-block or R(NAK) in case of PICC presence check Method 2 a) as described in ISO/IEC 14443-4:2018, 7.5.6.2. The response in step g) changes accordingly.

g) The LT sends a response using the frame format selected. Check if the answer from the LT is accepted by the PCD.

NOTE 2 Steps h) to j) may not be applicable when a PCD is embedded in a product.

h) The PCD shall send an S(DESELECT) request using the frame format selected.

i) The LT sends an S(DESELECT) response using the frame format selected. Check, if the answer from the PCD-test apparatus is accepted by the PCD.

j) The PCD shall start the polling sequence using short frame (for Type A) and standard frame (for Type B).

Table L.1 — Frame format bytes

Case	Supported frames																Indication that no framing options are supported	
	PCD to PICC (tag '80')								PICC to PCD (tag '81')								PCD to PICC (tag '82')[a]	PICC to PCD (tag '83')[b]
	b8	b7	b6	b5	b4	b3	b2	b1	b8	b7	b6	b5	b4	b3	b2	b1		
1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	No tag	No tag
2	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	No tag	Tag present, Length=0
3	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1	No tag	Tag present, Length=1, Value = '00'
4	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1	Tag present, Length=1, Value = '00'	Tag present, Length=1, Value = '00'
5	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	Tag present, Length=0	No tag
6	1	0	0	0	0	0	1	1	1	0	0	0	0	0	1	1	Tag present, Length=1, Value = '00'	No tag

[a] Tag '82' shall always be omitted for Type A PCD to PICC bit rates up to $f_c/16$ and higher than $f_c/2$.

[b] Tag '83' shall always be omitted for Type A.

Table L.2 — Framing options for Type A PCD to PICC bit rates greater than $f_c/16$ and less than $3f_c/4$

Case	Framing options PCD to PICC (tag '82')							
	b8	b7	b6	b5	b4	b3	b2	b1
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	1
9	0	0	0	0	0	0	1	0
10	0	0	0	0	0	1	0	0
11	0	0	0	0	0	1	1	1

Table L.3 — Framing options for Type B

Case	Framing options PCD to PICC (tag '82')								Framing options PICC to PCD (tag '83')							
	b8	b7	b6	b5	b4	b3	b2	b1	b8	b7	b6	b5	b4	b3	b2	b1
12	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
20	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
21	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0
22	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0
23	0	0	0	0	0	1	1	1	0	0	0	0	0	1	1	1

Table L.4 gives part of the procedure as a scenario.

Table L.4 — Scenario L.1: Frame format selection using S(PARAMETERS) blocks

PCD		LT
S(PARAMETERS)	→	
	←	S(PARAMETERS) frame format indication according to Table L.1, Table L.2 and Table L.3
S(PARAMETERS), frame format activation respecting the requirements defined in ISO/IEC 14443-4:2018, 10.5	→	
	←	S(PARAMETERS) frame format acknowledgement (using standard frame)
$I(0)_0$ (using activated PCD to PICC frame format and framing options)	→	
	←	$I(0)_0$ (using activated PICC to PCD frame format and framing options)
S(DESELECT) request (using activated PCD to PICC frame format and framing options)	→	
	←	S(DESELECT) response (using activated PICC to PCD frame format and framing options)
Polling sequence (using short/standard frame)	→	

L.1.1.4 Test report

Only if the PCD behaves according to Scenario L.1 in each test case, then the test result is PASS. In any other case, the test result is FAIL. The test report should document the frame formats and framing options chosen by the PCD in each direction and each test case.

L.1.2 PICC tests

L.1.2.1 Scope

This test is to verify the proper usage of frames with error correction including framing options.

These tests apply only to PICCs which support frames with error correction in at least one direction.

L.1.2.2 Apparatus

In these tests the PICC-test-apparatus shall be configurable to change the frame format during the test procedure. The PICC-test-apparatus shall be able to monitor the used frames on each stage of this test procedure.

L.1.2.3 Procedure

Apply the following procedure twice. Once for a bit rate of $f_c/128$ in each direction, and, if optional bit rates are supported, once with the maximum bit rate in each direction.

a) The PICC-test-apparatus activates the PICC according to G.5.1.2.

b) The PICC-test-apparatus sends an S(PARAMETERS) block to request frame format parameters.

c) The PICC shall answer with an S(PARAMETERS) block including frame format indication bytes.

d) The PICC-test-apparatus activates frame with error correction in PCD to PICC direction, if supported by the PICC, with no framing options using an S(PARAMETERS) block. If frame with error correction in PCD to PICC is not supported, continue with step m).

e) The PICC shall acknowledge the received S(PARAMETERS) block with an S(PARAMETERS) block response using standard frame format.

f) The PICC-test-apparatus sends $I(0)_0$ block using the frame format selected.

g) The PICC shall send a response using the frame format selected.

NOTE The PICC may answer with S(WTX) block. The PICC-test-apparatus needs to proceed accordingly.

h) The PICC-test-apparatus sends an S(DESELECT) request using the frame format selected.

i) The PICC shall send an S(DESELECT) response using the frame format selected.

j) The PICC-test-apparatus sends a WUPA for Type A in a short frame or WUPB for Type B in a standard frame.

k) The PICC shall answer with an ATQA when Type A or ATQB when Type B.

l) Repeat steps a) to k) for each of the following supported cases by the PICC.

1) In step d), the PICC-test-apparatus activates with SYNC suppression.

2) In step d), the PICC-test-apparatus activates with SOF/EOF suppression.

3) In step d), the PICC-test-apparatus activates with start bit and stop bit suppression.

4) In step d), the PICC-test-apparatus activates SYNC suppression and SOF/EOF suppression.

5) In step d), the PICC-test-apparatus activates SYNC suppression and start bit and stop bit suppression.

6) In step d), the PICC-test-apparatus activates SOF/EOF suppression and start bit and stop bit suppression.

m) Repeat steps a) to l) for each of the following supported cases by the PICC.

1) In step d), the PICC-test-apparatus activates frames with error correction for the PICC to PCD direction.

2) In step d), the PICC-test-apparatus activates frames with error correction for both directions.

Table L.5 gives part of the procedure as a scenario.

Table L.5 — Scenario L.2: Frame format selection using S(PARAMETERS) blocks

PICC-test-apparatus		PICC
S(PARAMETERS)	→	
	←	S(PARAMETERS) frame format indication
S(PARAMETERS) frame format activation	→	
	←	S(PARAMETERS) frame format acknowledgement (using standard frame)
$I(0)_0$ (using activated PCD to PICC frame format and framing options)	→	
	←	$I(0)_0$ (using activated PICC to PCD frame format and framing options)
S(DESELECT) (using activated PCD to PICC frame format and framing options)	→	
	←	S(DESELECT) (using activated PICC to PCD frame format and framing options)
WUPA (using short frame) or WUPB (using standard frame)	→	
	←	ATQA or ATQB (both are using standard frame)

L.1.2.4 Test report

Only if the PICC responds as indicated in the procedure, then the test result is PASS. In any other case, the test result is FAIL.

L.2 Error correction mechanism

L.2.1 PCD test

L.2.1.1 Scope

This test is to verify the capability of the PCD to correctly receive blocks with error correction for the following cases:

1) no bit error in each modified Hamming sub-block;

2) one bit error in each modified Hamming sub-block;

3) two bit errors in the first modified Hamming sub-block (to check CRC_32).

This test applies only to PCDs which support frames with error correction in PICC to PCD direction.

L.2.1.2 Apparatus

In this test the PCD-test-apparatus shall be configurable to change the frame format during the test procedure. The PCD-test-apparatus shall be able to monitor the used frames on each stage of this test procedure.

L.2.1.3 Procedure

Apply the following procedure.

a) The UT performs the protocol activation procedure according to H.1.9.2 for Type A or H.1.9.3 for Type B and the negotiation of the frame with error correction.

b) PCD shall send $I(0)_0$ block using the frame format selected. This block and subsequent PCD blocks may also be $I(1)_0$, S-block or R(NAK) in case of PICC presence check Method 2 a) as described in ISO/IEC 14443-4:2018, 7.5.6.2. The response in the following steps changes accordingly.

c) The LT sends an $I(0)_0$ response using the frame with error correction containing no bit error in each modified Hamming sub-block. Check, if the answer from the LT is accepted by the PCD.

d) PCD shall send $I(0)_1$ block using the frame format selected.

e) The LT sends an $I(0)_1$ response using the frame with error correction format and containing two bit errors in first modified Hamming sub-block.

f) PCD shall send a $R(NAK)_1$ block.

g) Repeat steps a) to f) with a single bit error in each modified Hamming sub-block in step c).

Table L.6 gives part of the procedure as a scenario.

Table L.6 — Scenario L.3: Exchange of frames with error correction

PCD		LT
$I(0)_0$	→	
	←	$I(0)_0$ (with no, or exactly one bit error in each modified Hamming sub-block)
$I(0)_1$	→	
	←	$I(0)_1$ (with two bit errors in the first modified Hamming sub-block)
$R(NAK)_1$	→	

L.2.1.4 Test report

Only if the PCD matches the expected Scenario L.3, then the test result is PASS. In any other case, the test result is FAIL.

L.2.2 PICC test

L.2.2.1 Scope

This test is to verify the capability of the PICC to correctly receive blocks with error correction for the following cases:

1) no bit error in each modified Hamming sub-block;

2) one bit error in each modified Hamming sub-block;

3) two bit errors in the first modified Hamming sub-block (to check CRC_32).

This test applies only to PICCs which support frames with error correction in PCD to PICC direction.

L.2.2.2 Apparatus

In this test the PICC-test-apparatus shall be used to emulate frames with error correction. The PICC-test-apparatus shall be able to monitor the used frames on each stage of this test procedure.

L.2.2.3 Procedure

Apply the following procedure.

a) The PICC-test-apparatus activates the PICC and frames with error correction at least in PCD to PICC direction and without framing options according to G.5.1.2.

b) The PICC-test-apparatus sends $I(0)_0$ block using the frame with error correction format containing no bit error in each modified Hamming sub-block.

c) The PICC shall send $I(0)_0$ response using the selected frame format.

d) The PICC-test-apparatus sends $I(0)_1$ block using the frame with error correction format containing two bit errors in first modified Hamming sub-block.

e) The PICC shall stay Mute.

f) Repeat steps a) to e) with a single bit error in each modified Hamming sub-block in step b).

Table L.7 gives part of the procedure as a scenario.

Table L.7 — Scenario L.4: Exchange of frames with error correction

PICC-test-apparatus		PICC
$I(0)_0$ (with no, or exactly one bit error in each modified Hamming sub-block)	→	
	←	$I(0)_0$
$I(0)_1$ (with two bit errors in the first Hamming-block)	→	
	←	Mute

L.2.2.4 Test report

Only if the PICC responds as indicated in the procedure, then the test result is PASS. In any other case, the test result is FAIL.

L.3 Logical operation of PICC when frame with error correction is activated

L.3.1 PICC reaction to ISO/IEC 14443-4 scenarios when frame with error correction is activated

L.3.1.1 Scope

This test is to determine the behavior of the PICC according to ISO/IEC 14443-4:2018, Clause 7, when the frame with error correction format as per ISO/IEC 14443-4:2018, Clause 10 has been activated. This test uses implementations of the protocol scenarios of ISO/IEC 14443-4:2018, Annex B.

This test applies only to PICCs which support frames with error correction in at least one direction.

L.3.1.2 Procedure

Perform the following steps for Scenario G.33, Scenario G.34, Scenario G.53 and Scenario G.54:

a) The PICC-test-apparatus activates the PICC according to G.5.1.2, use CID = 0.

b) The PICC-test-apparatus activates frames with error correction in all directions the PICC has indicated it supports without framing options. The PICC shall acknowledge the parameter activation.

c) For each step in each of the scenarios do:

   1) Send the command as described in the PICC-test-apparatus column,

   2) check that the response matches the one of the PICC column.

d) End for.

L.3.1.3 Test report

Only if the PICC responds as indicated in the procedure, then the test result is PASS. In any other case, the test result is FAIL.

L.3.2 Reduction of bit rate during chaining when frame with error correction is activated

L.3.2.1 Scope

This test is to verify the capability of the PICC to complete of a chain of transmitted frames using frame with error correction format when the bit rate is reduced by the PCD during chaining, as described in ISO/IEC 14443-4:2018, 7.5.1.

This test applies only to PICCs which support frames with error correction in at least one direction, and if such PICC supports activation of at least two bit rates in each direction by S(PARAMETERS).

L.3.2.2 Procedure

For each of the scenarios described in Table L.8 and Table L.9, perform the following steps:

a) The PICC-test-apparatus activates the PICC according to G.5.1.2, use CID = 0.

b) The PICC-test-apparatus activates frames with error correction in all directions the PICC has indicated it supports without framing options, and activates bit rates in both directions where at least one lower bit rate can be activated by S(PARAMETERS) exchange. The PICC shall acknowledge the parameter activation.

c) For each step in the scenario do:

1) Send the command as described in the PICC-test-apparatus column,

2) check that the response matches the one of the PICC column.

d) End for.

Table L.8 — Scenario L.5: Reduction of PCD to PICC bit rate during PCD chaining when frame with error correction is activated

Step	PICC-test-apparatus		PICC
1	$I(1)_0$(TEST_COMMAND1(3)$_1$)	→	
		←	$R(ACK)_0$
2	$I(1)_1$(TEST_COMMAND1(3)$_2$, ~CRC_32)	→	
		←	Mute
3	S(PARAMETERS), activation of a PCD to PICC bit rate lower than the current one[a]	→	
		←	S(PARAMETERS), acknowledge (using previous PCD to PICC bit rate)
4	$R(NAK)_1$ (using selected PCD to PICC bit rate)	→	
		←	$R(ACK)_0$
5	$I(1)_1$(TEST_COMMAND1(3)$_2$)	→	
		←	$R(ACK)_1$
6	$I(0)_0$(TEST_COMMAND1(3)$_3$)	→	
		←	$I(0)_0$(TEST_RESPONSE1(3))
7	$I(0)_1$(TEST_COMMAND1(1))	→	
		←	$I(0)_1$(TEST_RESPONSE(1))

[a] Example of function tags for reducing PCD to PICC bit rate to $f_c/32$: 'A0 06 A3 04 83 02 04 00'

Table L.9 — Scenario L.6: Reduction of PICC to PCD bit rate during PICC chaining when frame with error correction is activated

Step	PICC-test-apparatus		PICC
1	$I(0)_0$(TEST_COMMAND2(3))	→	
		←	$I(1)_0$(TEST_RESPONSE2(3)$_1$)
2	$R(ACK)_1$	→	
		←	$I(1)_1$(TEST_RESPONSE2(3)$_2$)
3	S(PARAMETERS), activation of a PICC to PCD bit rate lower than the current one[a]	→	
		←	S(PARAMETERS), acknowledge (using previous PICC to PCD bit rate)
4	$R(ACK)_1$ (using selected PICC to PCD bit rate)	→	
		←	$I(1)_1$(TEST_RESPONSE2(3)$_2$)
5	$R(ACK)_0$	→	
		←	$I(0)_0$(TEST_RESPONSE2(3)$_3$)
6	$I(0)_1$(TEST_COMMAND1(1))	→	
		←	$I(0)_1$(TEST_RESPONSE1(1))

[a] Example of function tags for reducing PICC to PCD bit rate to $f_c/32$: 'A0 06 A3 04 84 02 04 00'

L.3.2.3 Test report

Only if the PICC responds as indicated in the procedure, then the test result is PASS. In any other case, the test result is FAIL.

Annex M
(normative)

PCD phase stability analysis tool

M.1 Overview

The working principle of the PCD phase stability analysis tool is illustrated in Figure M.1.

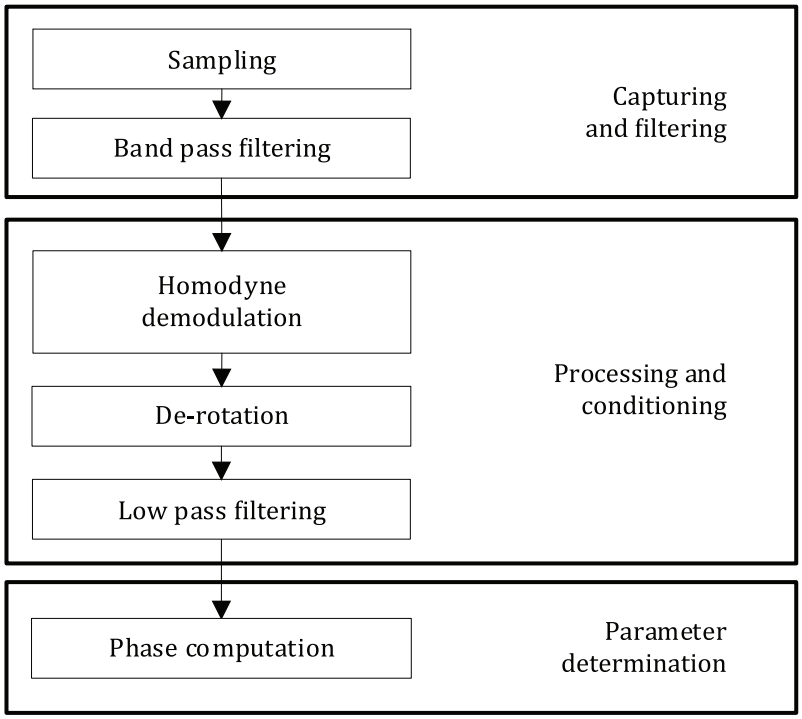


Figure M.1 — PCD phase stability analysis tool block diagram

M.2 Sampling

The signal acquiring device used for signal capturing shall fulfil the requirements defined in 5.2. The time and voltage data of the unmodulated carrier shall be transferred to a suitable computer software.

M.3 Band pass filtering

The filter of E.3.1 shall be used.

M.4 Homodyne demodulation

The filtered signal shall be demodulated using the homodyne demodulation.

M.5 De-rotation

The procedure defined in K.3.6 shall be used.

M.6 Low pass filtering

A 4th order, Butterworth type low pass filter with 3-dB cut off frequency at 500 kHz shall be used for filtering higher frequency components. The filter characteristic is illustrated in Figure M.2.

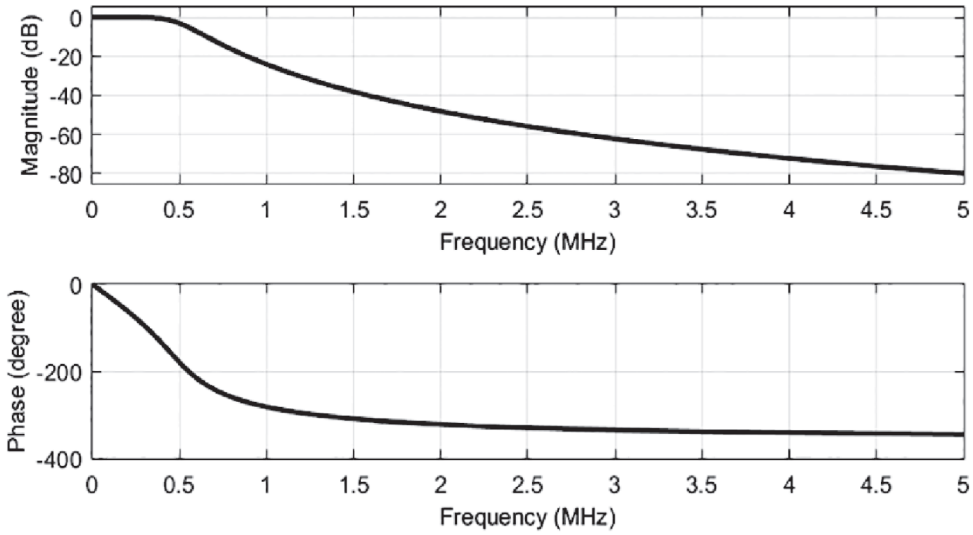


Figure M.2 — Filter characteristics

M.7 Phase computation

The instantaneous phase over time is computed by taking the argument of the complex signal. The maximum phase drift is the difference between the maximum absolute phase value and the minimum absolute phase value.

M.8 Program of the PCD phase stability analysis tool

M.8.1 General

The following program written in the high-level interpreted programming language GNU Octave gives an example for the implementation of the phase stability analysis tool. Alternatively, this program may be executed using MATLAB®. The PCD phase stability analysis tool consists of 3 different files which should be placed in the same folder.

M.8.2 main_phase_stability_tool.m

```
% main PCD phase stability analysis tool
% ------------------------------------------------------------------%
% This tool analyses the phase stability of the unmodulated PCD carrier
% signal.
% OUPUT of the tool:
%   The peak-peak phase drift requirements according to ISO/IEC 14443-2
%
% v1.1 from 2016-08: contains editorial updates, removal of unused code,
% alignment on latest versions of 14443-2 and 10373-6 AMDs

clear all
close all

fileName = 'PATH\NAME.csv'; % PCD signal file name

% Comment in case MATLAB® is used
pkg load signal % needed for the function "butter"
```

```
PLOT = 1; % flag enabling ('1') / disabling ('0') plot functionality

% Read data from 1st column of each file, skipping the first lines
delimiter = ','; % delimiter used to separate columns
headerLines = 5; % number of header lines to skip
csv_sig = dlmread(fileName, delimiter, headerLines, 0);

% PCD signal time and amplitude vectors
t = csv_sig(:, 1);
sig = csv_sig(:, 2);
N_sig = size(sig, 1);
fsamp = floor(1 / (t(2) - t(1))); % Sampling frequency


% Display File and data information
fprintf('File name: %s\r\n', fileName )
fprintf('Sampling frequency: %4f MS/s\r\n', fsamp*1e-6 )
fprintf('Signal length: %2.2f ms \r\n', (t(end)-t(1))*1e3)
fprintf('-------------------------\r\n');


[PD_pp, f_offset] = phase_stability(sig, t, fsamp, PLOT);

% Display results
fprintf('Frequency offset to 13.56 MHz: %4.2f Hz\r\n', f_offset)
fprintf('Maximum phase drift peak-peak: %3.2f degrees\r\n', PD_pp)
fprintf('-------------------------\r\n')
```

M.8.3 phase_stability.m

```
function [PD_pp, f_offset] = phase_stability(sig, t, fsamp, PLOT)

% function [PD_pp, f_offset] = phase_stability(sig, t, fsamp, PLOT)
%
% DESCRIPTION: Computes the continuous phase of an unmodulated 13.56 MHz
% PCD carrier signal according to ISO/IEC 10373-6:2016 Amd 3
%
% INPUT:
%       sig........vector of amplitude values (signal)
%       t..........vector of time indices
%       fsamp......sampling frequency [Hz]
%       PLOT.......flag enabling plot functionality
% OUTPUT:
%       PD_pp......computed peak-peak phase drift [degrees]
%       f_offset...computed frequency offset to fc [Hz]

fc = 13.56e6; % nominal carrier frequency

% backoff to remove fade in effects of filtering
offset = round(100 * fsamp / fc);

% subtract global mean
mu_sig = mean(sig);
sig = sig - mu_sig;

% output statistics
fprintf('Mean of signal amplitude: %E\r\n', mu_sig)
fprintf('Maximum of signal amplitude: %E\r\n', max(sig) )
fprintf('Minimum of signal amplitude: %E\r\n', min(sig) )
fprintf('-------------------------\r\n');

% output warning if sampling frequency is too low
if fsamp < 7 * fc
    warning('Sampling rate is lower than minimum of 7*fc!')
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 1) Band pass filtering

[b,a] = butter(2, [(fc - 5e6) (fc + 5e6)] / (fsamp / 2));
```

```
sig = filter(b,a,sig);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 2) Homodyne demodulation

% find first maximum of signal
max_idx = 1;
for k = 2 : round(fsamp/fc)
    if sig(k) > sig(max_idx)
        max_idx = k;
    end
end

% remove first samples to guarantee that the signals' phase and the I-demodulator
% signals are inphase (avoids problems when using low sampling rates)
sig = sig(max_idx : end);
sig = sig(:);
N = length(sig);
t = t(1 : N);
t = t(:);

% signal demodulation
bb_sig = sig .* (exp(1i * 2 * pi * fc * t));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 3) Derotation: compute mean frequency offset

stop_idx = length(bb_sig) - round(fsamp / fc); % stop index

% 1st frequency offset estimation
f_offset_1 = correct_f_error(bb_sig, 1, stop_idx, fsamp, fc);
% derotate complex bb signal by f_offset_1
bb_sig_corr = bb_sig .* exp(-1i * (2 * pi * f_offset_1) * t);

% 2nd frequency offset estimation to ensure accurate offset estimation
f_offset_2 = correct_f_error(bb_sig_corr, 1, stop_idx, fsamp, fc);
% derotate complex bb signal by f_offset_2
bb_sig_corr = bb_sig_corr .* exp(-1i * (2 * pi * f_offset_2) * t);

f_offset = f_offset_1 + f_offset_2;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 4) Low pass filtering with 4th order Butterworth type low pass filter
%    with 3-dB cut off frequency at 500 kHz

[b,a] = butter(4, 0.5e6 / (fsamp / 2)); % get filter coefficients
% filter signal
out = filter(b,a,real(bb_sig_corr)) + 1i*filter(b,a,imag(bb_sig_corr));
% remove filter fade-in transient from signal
out = out(length(b) + 1 + offset : end);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 5) Phase computation:
%    a) compute instantaneous phase (degrees)
%    b) compute and output frequency offset to fc,nom and
%       phase drift peak-peak

phi = unwrap(angle(out)) * 180 / pi;

max_phi = max(phi);
min_phi = min(phi);
PD_pp = abs(max_phi - min_phi); % phase drift peak-peak

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PLOT results

if PLOT
    t = t(1 : length(out)); % adjust time vector
    t = t * 1e3; % time in ms

    % plot magnitude and phase of complex bb signal
```

```
    figure('color', [1 1 1])

    bx(1) = subplot(211); plot(t, abs(out), 'k')
    ylabel('magnitude (V)')
    grid on
    bx(2) = subplot(212); plot(t, phi, 'k')
    xlabel('time (ms)')
    ylabel('phase (degree)')
    linkaxes(bx, 'x')
    grid on
    axis tight
end
```

M.8.4 correct_f_error.m

```
function f_err = correct_f_error(in, strt_idx, stop_idx, fsamp, fc)

% function f_err = correct_f_error(in, strt_idx, stop_idx, fsamp, fc)
%
% DESCRIPTION: performs an estimation of the frequency error resulting
%              in a constant phase drift ==> estimates the slope
% INPUT:
%       in........complex IQ input signal
%       strt_idx..analysis start index [samples]
%       stop_idx..analysis stop index [samples]
%       fsamp.....sampling frequency [Hz]
%       fc........carrier frequency [Hz]
% OUTPUT:
%       f_err.....estimated frequency error

in = in(:);

% perform IIR filtering to suppress high frequency components
% (cut-off frequency is set to 1 MHz)
[b,a] = butter(4, 1e6 / (fsamp / 2));
x = filter(b, a, real(in)) + 1i * filter(b, a, imag(in)); % filter signal
% remove filter fade-in transient from signal
in = x(max(length(b), round(30 * fsamp / fc) + 1 : end));
stop_idx = min(stop_idx, length(in));

in = in(strt_idx : stop_idx);
N = length(in);
t = (0 : 1 / fsamp : (N - 1) / fsamp)';

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

f_norm = EstimateCarrierFreq(in);
f_err = f_norm * fsamp;

function FcEst = EstimateCarrierFreq(in_bb_unmod)

% function FcEst = EstimateCarrierFreq(in_bb_unmod)
%
% DESCRIPTION: performs frequency error measurement
%
% INPUT:
%       in_bb_unmod...non-modulated complex BB signal
%
% OUTPUT:
%       FcEst...measured frequency offset normalize to sampling frequency

in_bb_unmod = in_bb_unmod(:);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 1) Rough phase drift estimation
in_bb_unmod1 = in_bb_unmod ./ abs(in_bb_unmod); % complex absolute value

% Totally remove amplitude fluctuations
IQJumps = diff(in_bb_unmod1); %phasor speed (instantaneous)
```

```
% Average of abs of differentiated phasor signal
IQjumpsMeanAbs = mean(abs(IQJumps));
FcEst = asin(IQjumpsMeanAbs / 2) / pi;

% Frequency rotation
N1 = length(in_bb_unmod);
derotatedPhasor = in_bb_unmod.*exp(-1i * 2 * pi * FcEst * (1 : N1).');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 2) Fine phase drift estimation

phaseSignalStep2 = unwrap(angle(derotatedPhasor));

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;
```

Annex N
(normative)

PICC amplitude and phase drift analysis tool

N.1 Overview

The working principle of the PICC amplitude and phase drift analysis tool is illustrated in Figure N.1.

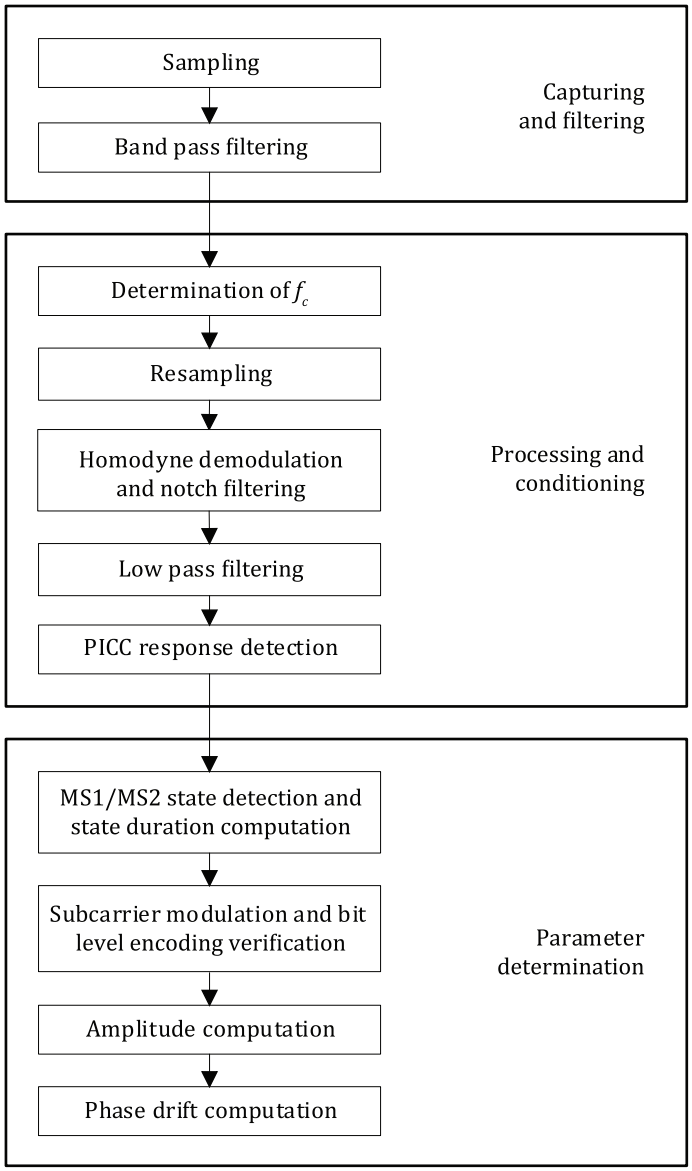


Figure N.1 — PICC amplitude and phase drift analysis tool block diagram

N.2 Sampling

The signal acquiring device used for signal capturing shall fulfil the requirements defined in 5.2. The time and voltage data of a complete PICC frame as captured by both the calibration coil and the load

modulation test circuit of Figure 2, with the conditions defined in 7.2.1.2 shall be transferred to a suitable computer software. Both, the calibration coil and the load modulation test circuit signals, shall be captured by the same digital sampling device as illustrated in Figure N.2.

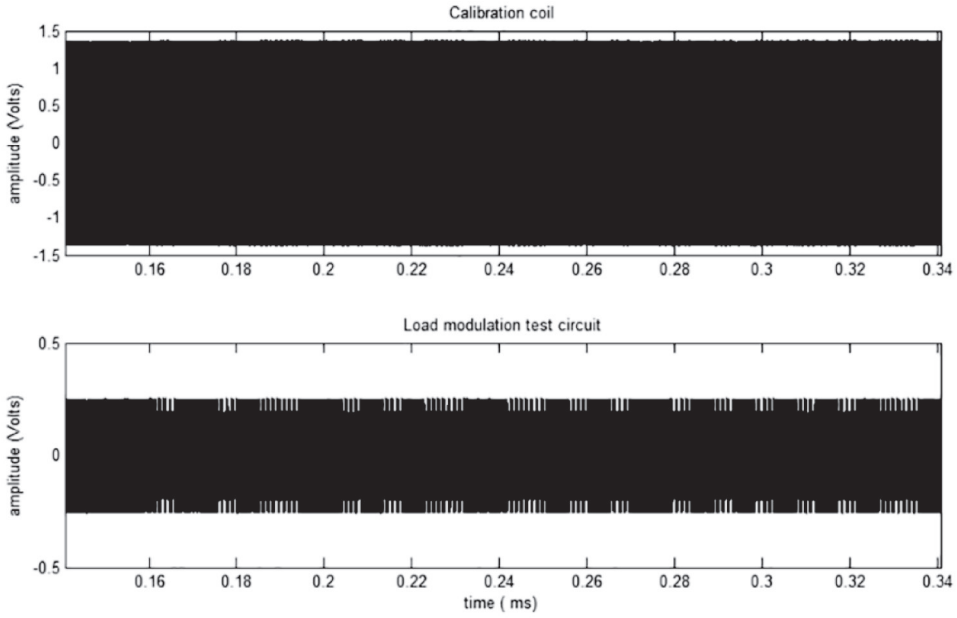


Figure N.2 — Example of a PICC response captured by both the calibration coil and the load modulation test circuit

N.3 Band pass filtering

The filter of E.3.2 shall be used.

N.4 Determination of f_c

The function defined in K.3.6 shall be used to determine f_c.

N.5 Resampling

The signal shall be resampled to an integer number multiple of f_c using linear interpolation. The integer number shall be at least 37.

N.6 Homodyne demodulation and notch-filtering

The filtered signals shall be demodulated using homodyne demodulation as shown in Figure N.3. The calibration coil signal shall be used to create demodulator signals. The real part of the Hilbert transformed calibration coil signal shall be used as In-phase demodulator signal and the imaginary part of the Hilbert transformed calibration coil signal shall be used as Quadrature-phase demodulator signal. The signal of the load modulation test circuit (also known as sense coils) is finally demodulated by the demodulator signals. In- and Quadrature- phase signals are filtered by a moving average type filter called Notch-Filter as specified in E.5.1. Finally, the filtered In- (Re{bb signal}) and Quadrature- (Im{bb signal}) phase signals represent the complex base band signal (bb signal).

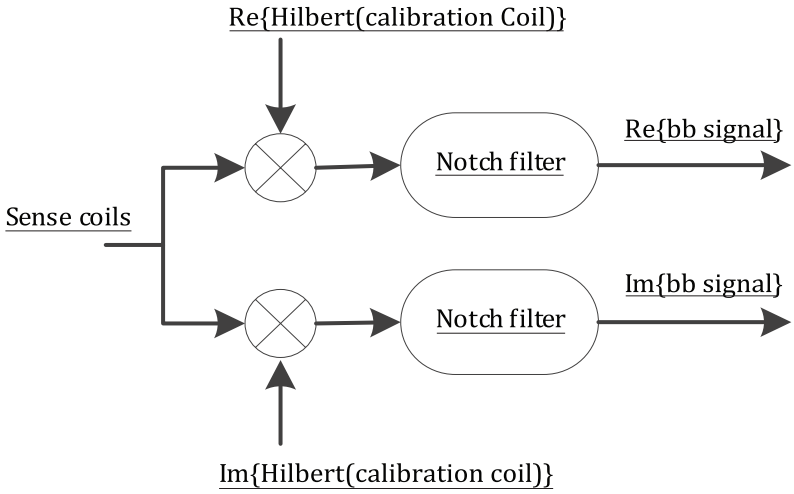


Figure N.3 — Demodulation

N.7 Low pass filtering

The analysis bandwidth of the complex base band signal shall be adjusted as a function of the subcarrier frequency using a 2nd order Butterworth type low-pass filter. The filter cut-off frequency is defined in Table N.1. The filter magnitude and phase responses are shown in Figure N.4.

Table N.1 — Butterworth filter cut-off frequency vs subcarrier frequency

Subcarrier frequency	Cut-off frequency
$f_c/16$	$3f_s$ (~2,54 MHz)
$f_c/8$	$3f_s$ (~5,09 MHz)
$f_c/4$	$2f_s$ (~6,78 MHz)
$f_c/2$	$2f_s$ (~13,56 MHz)

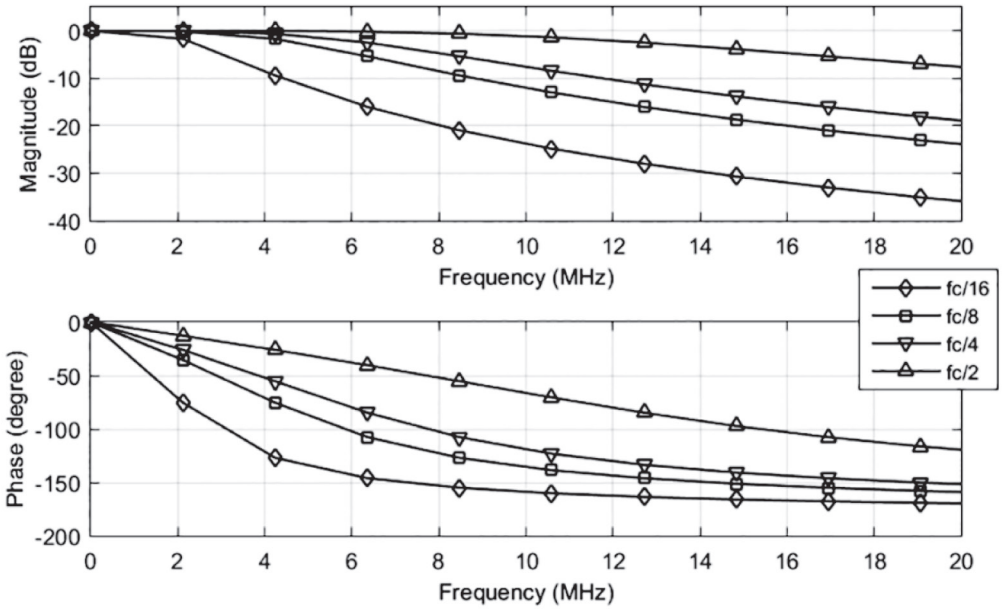


Figure N.4 — Subcarrier frequency dependent filtering

N.8 PICC response detection

A PICC response detector shall be implemented to determine the beginning and the end of the PICC response within the captured data stream, and to determine state US.

As an example, the program in N.12 determines the start and end of the PICC response following a two step approach:

a) Coarse start and end detection: Calculate the upper side band (USB) and lower side band (LSB) load modulation amplitudes versus time over the captured data stream. USB and LSB values are calculated every subcarrier period. The first significant change in amplitude (rising edge) corresponds to the start and the last significant change in amplitude (falling edge) to the end of the PICC response.

b) Accurate start and end detection: Compute the cross-correlation between the complex base band signal having higher power around the PICC response start and end with an ideal rectangular subcarrier period. The first local extrema correspond to the start of the PICC response. The last local extrema correspond to the end of the PICC response. The detected PICC response starting point is used to assign an initial subcarrier grid.

N.9 MS1/MS2 state detection and state duration computation

N.9.1 MS1/MS2 state detector

N.9.1.1 General

The MS1/MS2 state detector shall detect all modulations in the baseband PICC response and assign them correctly to either state MS1 or state MS2. The computations in the following subclauses shall be applied.

N.9.1.2 Signal power evaluation of the complex base band

The signal power of the In-phase and Q-phase part of the complex base band signal shall be individually computed and compared. If the signal power of one part is larger by a factor of 2.5 than the other part, the part with the smaller signal power is removed from further processing.

N.9.1.3 Correlation with an ideal rectangular subcarrier period during a phase transition

a) The cross-correlation between the complex base band signal components selected in N.9.1.2 and a signal representing a sequence of two ideal rectangular subcarrier periods during a phase transition shall be computed. The detected peaks contained in the cross-correlation result correspond to phase transitions due to bit changes in the PICC response.

b) Subcarrier grid re-alignment

   — In case of a PICC Type B response the subcarrier grid may be shifted by half a subcarrier period since the TR1 time may have a non-integer number length of subcarrier periods.

   — In case of a PICC Type A response with a bit rate higher than $f_c/128$ or a PICC Type B response with any bit rate a subcarrier grid fine adjustment shall be performed. Therefore, the difference of each bit change peak to the closest subcarrier grid shall be computed. The subcarrier grid is corrected by the average difference computed for all bit change occurrences.

N.9.1.4 Correlation with ideal rectangular subcarrier period

The cross-correlation between the complex base band signal components selected in N.9.1.2 and an ideal rectangular subcarrier period shall be computed. The detected peaks contained in the cross-correlation result correspond to the occurrence of a subcarrier period.

N.9.1.5 Phase transitions verification

The detected subcarrier periods shall be compared to the subcarrier grid of the PICC response computed in N.9.1.3.

NOTE If the average deviation of the phase shifts compared to the nominal position is more than ±10 % the program in N.12 interprets this as invalid input signal and terminates.

N.9.1.6 Symbol assignment

If all of the computations in N.9.1.2 to N.9.1.5 are completed successfully, MS1 and MS2 states shall be assigned to each subcarrier high and low state respectively. The state assignment shall be performed for the correlation signal of N.9.1.3 and of N.9.1.4 individually.

NOTE If the state assignments using signal of N.9.1.3 and signal of N.9.1.4 do not match, the program in N.12 terminates.

N.9.2 State duration

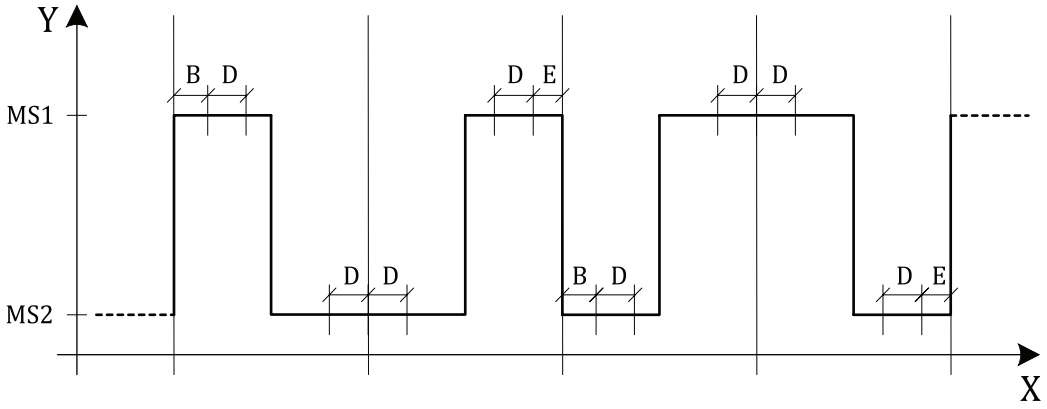
The state duration computation shall use the state information MS1 and MS2 of N.9.1. The state duration is defined as the stationary time duration of each occurrence of MS1 and MS2. This explicitly excludes the transitions between state occurrences. The beginning, end and duration of each occurrence of state MS1 and MS2 of the baseband PICC response are defined as follows and illustrated in Figure N.5.

1) The Parameter B shown in Figure N.5 defines an interval to exclude the part of a MS1/MS2 state occurrence after the start of a subcarrier period in case no phase transition occurred. Parameter B shall have a value of 17 % of the subcarrier period.

2) The Parameter E shown in Figure N.5 defines an interval to exclude the part of a MS1/MS2 state occurrence before the end of a subcarrier period in case no phase transition occurred. Parameter E shall have a value of 8 % of the subcarrier period.

3) The Parameter D shown in Figure N.5 defines the time duration of the MS1/MS2 state occurrence. Parameter D shall have a value of 20 % of the subcarrier period.

In case of a phase transition, Parameter 3 applies directly before the subcarrier end of the first subcarrier of the phase transition and applies directly after the subcarrier start of the subsequent subcarrier period.



Key

X Subcarrier periods
Y Amplitude
B Parameter B
D Parameter D
E Parameter E

Figure N.5 — State duration definition

N.10 Amplitude computation

For the amplitude computation the state and state duration information shall be used. The amplitude computation is split into two parts. The first part computes state US. The second part computes the magnitude of the complex vector difference between state US and each sample of each occurrence of state MS1. The procedure below shall be followed:

a) Compute the mean of the signal samples related to state US during a duration of $64/f_c$ before the PICC response starts. The last carrier period for US computation occurs two carrier periods before the first modulation edge of the PICC response.

b) Compute the magnitude of the complex vector difference between the mean US value of step a) and each sample of each occurrence of state MS1, as defined in ISO/IEC 14443-2:2020, 8.2.2.2.

N.11 Phase drift computation

The phase drift parameters $\varnothing_{\mathrm{LM,\ INTER}}$ and $\varnothing_{\mathrm{LM,\ INTRA}}$ shall be computed as defined in ISO/IEC 14443-2:2020, 8.2.2.3.

N.12 Program of the amplitude and phase drift analysis tool

N.12.1 General

The following program written in the high-level interpreted programming language GNU Octave gives an example for the implementation of the phase drift analysis tool. Alternatively this code program may be executed using MATLAB®.

N.12.2 readme.txt

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% README for the phase drift analysis tool v3.0
%
```

```
% 1) This tool analysis characteristics of a complete PICC response. The
%    captured data must contain the complete PICC response plus at least
%    200 carrier periods before and 20 after the PICC response.
%
% 2) TOOL input: The tool processes both, the captured calibration coil
%    And sense coils signals.
%
% 3) Open the file "main_PDA_tool.m" in MATLAB or OCTAVE and change the
%    Name and location of the file(s)
%
% Note: The main function of the tool expects 2 files, one for the
% calibration coil signal and one for the sense coils signal. If your
% oscilloscope supports to save 2 channels in 1 file, the main file
% has to be adapted.
%
% 4) Supported File format: CSV, MAT and TRC
%    a) 1st column time or variable "t" for mat-file
%    b) 2nd column amplitude of variable "sig" for mat-file
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

N.12.3 main_PDA_tool.m

```
% main phase drift analysis tool v 3.0
% ------------------------------------------------------------------- %
% This tool analysis a complete PICC response. For processing, min 200
% unmodulated carrier periods before and at least 20 after the PICC
% response are required.
% OUPUT of the tool:
% A) The 4 requirements acc. to ISO/IEC 14443-2:
%   1) Intrastate phase drift requirement
%   2) Interstate phase drift requirement
%   3) Magnitude of state MS1 requirement
%   4) Optional parameter of initial phase
% B) Plots and stores result figures:
%   Magnitude of state MS1 and MS2 over time + limit
%   Phase of MS1-MS2 signal over time + limits
%   Constellation diagram of time continuous PICC response + states MS1 &
%   MS2
%
% v3.0 from 2019-03:
%       - requires to specify the interface type and bit rate!
%       - start- and end of response detector modified to be much more
%         accurate; MS1/MS2/US state determination algorithm changed;
%         decoding of the PICC response added;
%
% Script has been tested with:
%    - Matlab R2015a
%    - Octave 4.4.0 with "signal" package version 1.3.2 and
%      "nan" package version 3.1.4

clear all
close all
clc

% ------------------------------------------- %
% Specify Interface Type, Bit rate and PICC class

% Interface Type
InterfaceT = 'Type_A'; % either 'Type_A' or 'Type_B'
% Bit Rate
bitRate = '106'; % either '106','212','424','848','1695', '3390' or '6780'
PICC_class = 1; % enter PICC class (1-6)

data_path = 'C:\...\...\';
fileName_calCoil = 'C1_XX.csv';  % Calibration coil signal file name
fileName_HHB = 'C2_XX.csv'; % Sense coils signal file name
% ------------------------------------------- %

% Flag to automatically switch between Octave and MATLAB
global octave_flag;
```

```
octave_flag = exist('OCTAVE_VERSION','builtin');
global DEBUG;
DEBUG = 0;

if octave_flag
  pkg load signal % Needed for the function "butter" and "findpeaks"
  pkg load nan % Needed for the function "rms"
end

% Define PICC Parameters
PICC_signal_encoding = struct('interface_type', InterfaceT, ...
                              'bitrate', bitRate);

encoding_parameters = get_encoding_parameters(PICC_signal_encoding);
fc = 13.56e6; % nominal carrier frequency [Hz]

% number of carrier periods per subcarrier period
f_sub_div = encoding_parameters.subcarrier_divider;

% Number of subcarrier cycles in one bit duration
etu_cyc = encoding_parameters.subcarrier_cycles_per_etu;


fprintf('------------ Input data ------------\r\n')
fprintf('Calibration coil signal source file: %s\r\n', fileName_calCoil)
fprintf('Sense coils signal source file: %s\r\n', fileName_HHB)
fprintf('Nominal carrier frequency fc: %4.2f MHz\r\n', fc * 10^(-6))
fprintf(['-) Communication signal interface: %s \n', ...
    '-) Bit rate: %s kbit/s\n\r'], ...
    PICC_signal_encoding.interface_type, PICC_signal_encoding.bitrate)

fprintf('Nominal subcarrier frequency: fc/%d (%4.1f kHz)\n\r', ...
    f_sub_div,  (fc / f_sub_div) * 10^(-3))
fprintf('PICC class: %d\r\n', PICC_class)

% check supported file formats
if strcmp(fileName_HHB(end-2:end), 'CSV') || ...
        strcmp(fileName_HHB(end-2:end), 'csv')
    data_source = 'CSV';
elseif strcmp(fileName_HHB(end-2:end), 'MAT') || ...
        strcmp(fileName_HHB(end-2:end), 'mat')
    data_source = 'MAT';
else
    error('No valid input file format selected!')
end


% read input data
if strcmp(data_source, 'CSV')
    delimiter = ','; % delimiter used to separate columns
    headerLines = 5; % number of header lines to skip
    % Read data from 1st column of each file, skipping the first lines
    csv_sigC1 = dlmread([data_path, fileName_calCoil], delimiter, ...
        headerLines, 0);
    csv_sigC2 = dlmread([data_path, fileName_HHB], delimiter, ...
        headerLines, 0);

elseif strcmp(data_source, 'MAT')
    load([data_path,fileName_calCoil], 't', 'sig');
    t = t(:);
    sig = sig(:);
    csv_sigC1 = [t, sig];
    load([data_path, fileName_HHB], 't', 'sig');
    t = t(:);
    sig = sig(:);
    csv_sigC2 = [t, sig];
    clear t;
end
```

```
% Calibration coil time and amplitude vectors
t1 = csv_sigC1(:, 1);
signal.sigC1 = csv_sigC1(:, 2);
N_sigC1 = size(signal.sigC1, 1);
fs1 = floor(1 / (t1(2) - t1(1)));
clear csv_sigC1
% Sense coils (Helmholtz bridge) time and amplitude vectors
t2 = csv_sigC2(:, 1);
signal.sigC2 = csv_sigC2(:, 2);
N_sigC2 = size(signal.sigC2, 1);
fs2 = floor(1 / (t2(2) - t2(1)));
clear csv_sigC2

fprintf('------------ Preprocessing ------------\r\n')
if abs(fs1 - fs2) * 1000 > fs1
    error('Sampling rates of both signals do not match!\r\n')
else
    fsamp = fs1;
end

fprintf('Input sampling frequency: %4.2f MS/s\r\n', fsamp*1e-6)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Signal analysis
date_txt = datestr(now, 29);
param.name = [data_path, date_txt, '_', fileName_calCoil(1 : end - 4)];
param.fsamp = fsamp;
param.fc = fc;
param.f_sub_div = f_sub_div;
param.PICC_class = PICC_class;
param.encoding = PICC_signal_encoding;

fs_old = fsamp;
fprintf('------------ Processing and Conditioning ------------\r\n')
[bb_sig_filt, hhb_sig, fc, fsamp] = ...
    processing_and_conditioning(signal, param);

signal.bb_sig_filt = bb_sig_filt;

f_sub_old = fc / 16;
spfs = round(fs_old / f_sub_old);

clear bb_sig_filt

param.fsamp = fsamp; % update structure
param.fc = fc; % update structure

[start_idx, end_idx, fft_usb, fft_lsb, fft_step, ret_val] = ...
    sliding_lma(hhb_sig, signal.bb_sig_filt, fc, fc / f_sub_div, ...
    fsamp, PICC_signal_encoding);
if ret_val ~= 0
    return
end

fprintf('------------ Subcarrier analysis ------------\r\n')
[subcarrier_states, bit_states_A_106, PD_struct, subcarrier_indices, ...
    ignore_last_subcarrier, ret_val] = ...
    subcarrier_analysis( signal.bb_sig_filt, start_idx, end_idx, ...
    param, signal.sigC1);

if ret_val ~= 0
    return
end

signal.subcarrier_indices = subcarrier_indices;


% ------------ Signal decoding ------------ %
[bit_states_B_A_not_106, SOF_start, ret_val] = signal_decoding( ...
    subcarrier_states, PICC_signal_encoding, ignore_last_subcarrier);
```

```
if ret_val ~= 0
    return
end

if strcmp(PICC_signal_encoding.interface_type, 'Type_A') && ...
        strcmp(PICC_signal_encoding.bitrate, '106')
    bit_states = bit_states_A_106;
else
    bit_states = bit_states_B_A_not_106;
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Print results
fprintf('------------ Results -------------\r\n')

%----- Print binary data contains in frame
padding_length = 80 - mod(length(bit_states), 80);
binary_data_str = sprintf('%d', bit_states);
binary_data_str = [binary_data_str, repmat(' ', 1, padding_length)];
binary_data_str = reshape(binary_data_str, 80, []);
binary_data_str = binary_data_str';
[number_of_rows, ~] = size(binary_data_str);

fprintf('Binary data contained in the frame:\n')

for k = 1 : number_of_rows
    fprintf('\t%s\n', binary_data_str(k, :));
end
fprintf('\n\r');

%----- Compute LMA
[usb_min, lsb_min, VLMA_max] = LMA_over_time(fft_usb, fft_lsb, ...
    fft_step, subcarrier_indices, bit_states, SOF_start, ...
    PICC_signal_encoding, ignore_last_subcarrier);

fprintf('Measured field strength: %3.2f A/m(rms)\r\n', PD_struct.h_field)
fprintf('------------ PICC load modulation (normative) -------------\r\n')
fprintf(['Minimum V_LMA: LSB: %2.2f mVp; USB: %2.2f mVp (minimum ', ...
     'limit: %1.2f mVp)\r\n'], ...
     lsb_min, usb_min, PD_struct.V_lma_min)
 fprintf('Maximum V_LMA: %2.2f mVp (maximum limit: %1.2f mVp)\r\n', ...
    VLMA_max, PD_struct.V_lma_max)
fprintf('Minimum modulus of MS1 - US: ')
fprintf('%2.2f mV (minimum limit: %1.2f mV)\r\n', ...
    PD_struct.min_mag_val, PD_struct.V_lma_min)
fprintf(['Intrastate phase drift (peak-peak): %2.2f degrees ', ...
    '(maximum limit: %1.2f degrees)\r\n'], ...
    PD_struct.max_intraState_PD, 40)
fprintf(['Interstate phase drift (peak-peak): %2.2f degrees ', ...
    '(maximum limit: %1.2f degrees)\r\n'], ...
    PD_struct.max_interState_PD, 30)
fprintf(['-----------------------------------------------------', ...
'---------\r\n'])
```

N.12.4 processing_and_conditioning.m

```
function [bb_sig_filt, hhb_sig, fc, fsamp] = ...
    processing_and_conditioning(signal, ...
    param)

% function [bb_sig_filt, hhb_sig, fc, fsamp] = ...
%   processing_and_conditioning(signal, ...
%   param)
%
% DESCRIPTION: Function for performing all steps from the 'Processing and
% conditioning' part
%
% INPUT:
%       signal.sigC1  ....time domain input signal from calibration coil
```

```
%       signal.sigC2  ....time domain input signal from sense coils
%       param.fsamp   ....sampling frequency [Hz]
%       param.f_sub_div...number of carrier cycles per subcarrier period
%       param.fc      ....carrier frequency [Hz]
%       param.dataRate....bit rate of modulated carrier [kbit/s]
%       param.PICC_class..defines the PICC class: {1, 2, 3, 4, 5, 6}
%       param.encoding....defines the encoding of the signal.
% OUTPUT:
%       bb_sig_filt...filtered complex baseband signal
%       hhb_sig...sense coils signal (filetered and aligned)
%       fc............Determined carrier frequency
%       fsame.........Samplig rate after resampling

% Change to save all figures as fig and emf in the directory of the data
SAVE = 0;
PLOT = 0;

if SAVE
    fprintf('Save functionality is enabled!\r\n');
    store_name = param.name;
end

cal_coil_sig = signal.sigC1;
cal_coil_sig = cal_coil_sig / max(abs(cal_coil_sig)); % normalize to +/-1

hhb_sig = signal.sigC2;

fsamp = param.fsamp;
fc = param.fc;

cal_coil_sig = cal_coil_sig(:);
hhb_sig = hhb_sig(:);

N_cc = length(cal_coil_sig);
N_hhb = length(hhb_sig);

if N_cc > N_hhb % if totally different data in the 2 files
    cal_coil_sig = cal_coil_sig(1 : N_hhb);
else
    hhb_sig = hhb_sig(1 : N_cc);
end


%>---------------------Band pass filtering---------------------------------

cutOff_freq = 7.5E6; %15MHz bandwidth -> fc +- 7.5Mhz
Wn = [fc - cutOff_freq, fc + cutOff_freq] / (fsamp/2);
[b, a] = butter(2, Wn); % Standard: 4th oder -> Octave docu: order 2n for
%  every filter except lowpass

cal_coil_sig = filter(b, a, cal_coil_sig);
hhb_sig = filter(b, a, hhb_sig);


%>-------------------------Determination of fc-----------------------------
[fc, fsamp, fsamp_int, fact] = fc_determination(cal_coil_sig, fsamp, fc);

%>-------------------------Resampling--------------------------------------
[fsamp, hhb_sig, cal_coil_sig] = resampling(hhb_sig, cal_coil_sig, ...
    fsamp, fsamp_int, fact);

samples_per_carrier = round(fsamp / fc);

%>-------------Homodyne demodulation and notch filtering------------------
[cal_coil_sig, hhb_sig, bb_sig_filt] = demodulation_and_filtering( ...
    hhb_sig, cal_coil_sig, samples_per_carrier );

%>----------------------Low pass filtering-------------------------------
[bb_sig_filt, index_correction] = low_pass_filtering(bb_sig_filt, ...
    fsamp, fc, param.f_sub_div);
hhb_sig = hhb_sig(index_correction : end);
```

```
cal_coil_sig = cal_coil_sig(index_correction : end);

if PLOT
    p1 = subplot(311);
    plot(cal_coil_sig);
    grid minor
    xlabel('Samples')
    ylabel('Signal in V')
    title('Calibration coil signal')


    p2 = subplot(312);
    plot(real(bb_sig_filt));
    grid minor
    xlabel('Samples')
    ylabel('Signal in V')
    title('real(Baseband signal)')

    p3 = subplot(313);
    plot(imag(bb_sig_filt));
    grid minor
    xlabel('Samples')
    ylabel('Signal in V')
    title('imag(Baseband signal)')

    linkaxes([p1, p2, p3], 'x')

end

end

function [fc, fsamp, fsamp_int, fact] = ...
    fc_determination(cal_coil_sig, fsamp, fc)

% function [fc, fsamp, fsamp_int, fact] = ...
%    fc_determination(cal_coil_sig, fsamp, fc)
%
% DESCRIPTION: Function for determining the frequency error between the
%              nominal carrier frequency (13.56 MHz) and the actual
%              carrier frequency from the calibration coil signal
%
% INPUT:
%       cal_coil_sig.....Signal from the calibration coil
%       fsamp.....Sampling frequency
%       fc.....Nominal carrier frequency
% OUTPUT:
%      fc.....Actual carrier frequency
%      fsamp.....Sampling frequency
%      fsamp_int.....Sampling frequency for up-sampling (integer number of
%                    actual fc)
%      fact.....fsamp_int / fsamp


n_fc = (fsamp / fc); % number of fc in sampling frequency fsamp

if n_fc <= 36 % ISO/IEC 10373-6 defines 500 MSample/s minimum
    fprintf(['Minimum sampling frequency requirement as defined in ', ...
        'ISO/IEC 10373-6 is not met!\r\n'])
    fprintf('Minimum Sampling frequency is 500MSpS. ')
    fprintf('Actual Sampling frequency is: %3.0f MSpS \r\n', n_fc*fc*1e-6)
    fprintf('Program terminates!\r\n')
    PD_struct = [];
    bb_sig_filt = [];
    return
elseif n_fc > 36 && n_fc < 74
    n_fc = ceil(n_fc);
else
    n_fc = 74;
end

N_carrier = 64; % number of unmodulated carrier periods;
```

```
n_silence = ceil(fsamp / fc * N_carrier);
t = (0 : n_silence - 1) / fsamp;

% Estimate Test PCD assembly generator frequency fc offset to 13.56 MHz
bb_sig_calCoil = hilbert(cal_coil_sig(1 : n_silence));
bb_sig_calCoil = bb_sig_calCoil .* exp(-1i * (2 * pi * fc) * t');

f_err = 0;
for n= 1:3
    f_e = correct_f_error(bb_sig_calCoil, 1, n_silence, fsamp, fc);
    bb_sig_calCoil = bb_sig_calCoil .* exp(-1i * (2 * pi * f_e) * t');
    f_err = f_err + f_e;
end

fprintf('Carrier frequency offset to 13.56 MHz: %1.0f Hz\r\n', f_err)

fc = fc + f_err;
fsamp_int = n_fc * fc;
fact = fsamp_int / fsamp;

end

function [cal_coil_sig, hhb_sig, bb_sig_filt] = ...
    demodulation_and_filtering(hhb_sig, cal_coil_sig, samples_per_carrier)

% function [cal_coil_sig, hhb_sig, bb_sig_filt] = ...
%   demodulation_and_filtering(hhb_sig, cal_coil_sig, samples_per_carrier)
%
% DESCRIPTION: Function for demodulating and notch-filtering the signal
%              HF-signal of the sense coils as described in
%              ISO/IEC 10373-6:2018 Annex N
%
% INPUT:
%       hhb_sig.................The HF-signal from the sense coils
%       cal_coil_sig............The calibration coil signal
%       samples_per_carrier.....Samples in one carries
% OUTPUT:
%      fc............Actual carrier frequency
%      fsamp.........Sampling frequency
%      fsamp_int....Sampling frequency for up-sampling (integer number of
%       actual fc)
%      fact.........fsamp_int / fsamp

% Preprocessing to make a periodic signal
[strt_idx, stp_idx] = make_periodic_frame(cal_coil_sig, ...
    samples_per_carrier);

cal_coil_sig = cal_coil_sig(strt_idx : stp_idx);
hhb_sig = hhb_sig(strt_idx : stp_idx);

% a) make demodulation signal (90deg shifted version)
cal_coil_90 = -imag(hilbert(cal_coil_sig));
cal_coil_90 = cal_coil_90 / max(abs(cal_coil_90));
cal_coil_sig = cal_coil_sig / max(abs(cal_coil_sig));
% b) Demodulation
bb_sig = 2*(hhb_sig .* cal_coil_sig + 1i * hhb_sig .* cal_coil_90);

% define length of moving average filter
filter_offset = 3 * samples_per_carrier;
IQFilt = ones(1, samples_per_carrier) / samples_per_carrier;

%>----------------------------Filtering-----------------------------------
bb_sig_filt = filter(IQFilt, 1, ...
    real(bb_sig)) + 1i * filter(IQFilt, 1, imag(bb_sig));

bb_sig_filt = bb_sig_filt((samples_per_carrier + filter_offset + 1) : ...
    end);
hhb_sig = hhb_sig((samples_per_carrier + filter_offset + 1) : end);
cal_coil_sig = cal_coil_sig((samples_per_carrier + filter_offset + 1) :...
    end);
end
```

```
function [fsamp, hhb_sig, cal_coil_sig] = resampling(hhb_sig, ...
    cal_coil_sig, fsamp, fsamp_int, fact)

% function [fsamp, hhb_sig, cal_coil_sig] = resampling(hhb_sig, ...
%   cal_coil_sig, fsamp, fsamp_int, fact)
%
% DESCRIPTION: Function for performing the upsampling of the captured
%              signals to an integer number of the carrier frequency as
%              described in ISO/IEC 10373-6:2018 Annex N
%
% INPUT:
%       hhb_sig.....The HF-signal from the sense coils
%       cal_coil_sig.....The calibration coil signal
%       samples_per_carrier.....Samples in one carries
%       fsamp.....Current sampling frequency
%       fsamp_int.....Sampling frequency after upsampling
%       fact.....fsamp_int / fsamp
% OUTPUT:
%      fsamp.....Sampling frequency after upsampling
%      hhb_sig.....Upsampled and interpolated sense coil signal
%      cal_coil_sig.....Upsampled and interpolated calibration coil signal
%
%

LEN = length(hhb_sig);
t = (0 : (LEN - 1))' / fsamp;
t_int = (0 : ceil(fact * (LEN - 1)))' / fsamp_int;

cal_coil_sig = interp1(t, cal_coil_sig, t_int, 'linear');
hhb_sig = interp1(t, hhb_sig, t_int, 'linear');
% check if no "NaN" values are in signal
check_Nan_cc = isnan(cal_coil_sig);
if check_Nan_cc
    idx = find(check_Nan_cc == 1);
    cal_coil_sig(idx) = 0;
end
check_Nan_hhb = isnan(cal_coil_sig);
if check_Nan_hhb
    idx = find(check_Nan_hhb == 1);
    hhb_sig(idx) = 0;
end

fsamp = fsamp_int; % assign new sampling frequency to fsamp
fprintf('Resampling rate used for processing: %3.2f MS/s \r\n', ...
    fsamp * 1e-6)


end


function [bb_sig_filt_lp, index_correction] = low_pass_filtering(...
    bb_sig_filt, fsamp, fc, subcarrier_divider)

% function [bb_sig_filt_lp, index_correction] = low_pass_filtering(...
%    bb_sig_filt, fsamp, fc, subcarrier_divider)
%
% DESCRIPTION: Function for low-pass filtering the demodulated and
%              notch-filtered baseband signal as described in
%              ISO/IEC 10373-6:2018 Annex N
%
% INPUT:
%       bb_sig_filt.....Notch-filtered baseband signal
%       fsamp.....Sampling frequency
%       fc.....Carrier frequency
%       subcarrier_divider.....Factor between the carrier- and the
%                              subcarrier frequency (Bitrate dependent)
% OUTPUT:
%      bb_sig_filt_lp.....The low-pass filtered baseband signal
%      index_correction.....Samples to be excluded from the sense-coil-
%                           and calibration-coil-signal for a proper
```

```
%                           alignment of the signals
%


fsub = fc / subcarrier_divider; % subcarrier frequency
sp_cw = fsamp / fc; % samples per carrier

if subcarrier_divider >= 8
    Wn_lp = 3 * fsub / (fsamp / 2);
else
    Wn_lp = 2 * fsub / (fsamp / 2);
end

[b_lp, a_lp] = butter(2, Wn_lp); % 2nd order Butterworth low-pass filter
bb_sig_filt_lp = filter(b_lp, a_lp, bb_sig_filt);

N_signal = length(bb_sig_filt);

% align filtered signal to original signal
a_re = xcov(real(bb_sig_filt_lp), (real(bb_sig_filt)));
a_re = a_re(N_signal : end);
[v_re, i_re] = max(a_re);
%clear a_re; % clean up
a_im = xcov(imag(bb_sig_filt_lp), (imag(bb_sig_filt)));
a_im = a_im(N_signal : end);
[v_im, i_im] = max(a_im);
%clear a_im; % clean up

% select channel with higher signal energy
if v_re >= v_im
   filt_delay = i_re;
else
   filt_delay = i_im;
end


index_correction = 10 * sp_cw;
bb_sig_filt_lp = bb_sig_filt_lp(index_correction + filt_delay : end);
end
```

N.12.5 sliding_lma.m

```
function [start_idx, end_idx, usb, lsb, fft_step, ret_val] = ...
    sliding_lma(hf_sig, bb_sig, fc, fsub, fsamp, encoding)

% function [start_idx, end_idx, usb, lsb, fft_step, ret_val] = ...
%    sliding_lma(hf_sig, bb_sig, fc, fsub, fsamp, encoding)
%
% DESCRIPTION: Function for determining the start and end of the PICC
%              response.
%
%              The function calculates the sliding DFT as defined in
%              ISO/IEC 10373-6 Annex F.
%              The DFT of 6 subcarrier cycle long window of the HF signal
%              is computed at the frequencies fc +- fsub. A bartlett
%              window is used for windowing. The step width is 1
%              subcarrier cycle.
%
%              The approximate start and end indices are determined via
%              the first rising and last falling edge of the amplitude.
%              After that the exact start and end are determines via
%              correlation with one subcarrier cycle
%
%
% INPUT:
%        hf_sig...sense coils signal aligned to the filtered complex
%                 baseband signal
%        bb_sig...Complex baseband signal
%        fc.......Carrier frequency (Hz)
%        fsub.....Subcarrier frequency (Hz)
```

```
%        fsamp....Sampling frequency (Hz)
%        encoding...struct containing interface type and bitrate
%
% OUTPUT:
%       start_idx......Start of the PICC response
%       end_idx........End of the PICC response
%       fft_usb........Amplitude of the upper sideband (mV)
%       fft_lsb........Amplitude of the lower sideband (mV)


% Enables to plot the LMA over the PICC response
EXTENDED_PLOT = 0;
global DEBUG;
% Enables to plot the determined start and end of the PICC response
PLOT = EXTENDED_PLOT | 0;

% Parameter calculation
f_usb = fc + fsub;
f_lsb = fc - fsub;
samples_per_subcarrier_cycle = round(fsamp / fsub);
samples_per_carrier = round(fsamp / fc);

ret_val = 0;
end_idx = -1;

% ---------------------- DFT calculation ----------------------------- %

% Length of one subcarrier cycle is bitrate dependent
switch encoding.bitrate
    case {'106', '212', '424', '848'}
        fft_step = samples_per_carrier * 16;
    case '1695'
        fft_step = round(samples_per_carrier * 8);
    case '3390'
        fft_step = round(samples_per_carrier * 4);
    case '6780'
        fft_step = round(samples_per_carrier * 2);
end

N = length(hf_sig);
% Length of the DFT: 6 subcarrier cycles
fft_length = 6 * samples_per_subcarrier_cycle;
% Start indices of each frame
fft_start = 1 : fft_step : (N - fft_length);

window = bartlett(6 * samples_per_subcarrier_cycle);
cf = 2;

n = 0 : 1 : fft_length - 1;
n = n(:);

% Scaled fft vectors for each sideband
fft_usb = cf * window / fft_length .* exp(1i * 2 * pi * n * f_usb / ...
    fsamp);
fft_lsb = cf * window / fft_length .* exp(1i * 2 * pi * n * f_lsb / ...
    fsamp);

fft_usb = fft_usb * 1000; % convert to mV
fft_lsb = fft_lsb * 1000; % convert to mV

usb = zeros(length(fft_start), 1);
lsb = zeros(length(fft_start), 1);

for k = 1 : length(fft_start)
    usb(k) = hf_sig(fft_start(k) : fft_start(k) + fft_length - 1).' * ...
        fft_usb;
    lsb(k) = hf_sig(fft_start(k) : fft_start(k) + fft_length - 1).' * ...
        fft_lsb;
end

if EXTENDED_PLOT
```

```
    fprintf(['plot functionality in sliding_lma() enabled. This ', ...
        'will generate plots with additional information about how', ...
        'the start and end of the PICC response are found.\n', ...
        'This also enables the regular plot functionality.\n\r'])

    figure()
    fprintf(['\tFigure %d shows the amplitude of the upper and ', ...
        'lower sidebands of the sliding DFT and the corresponding ', ...
        'baseband and HF signal.\n\r'], get(gcf, 'Number'))

    p1 = subplot(211);
    plot((0 : length(usb) - 1) * fft_step + 1, abs(usb) )
    hold on
    plot((0 : length(usb) - 1) * fft_step + 1, abs(lsb) )
    legend('usb', 'lsb')
    title('Amplitudes of running DFT for USB and LSB')
    grid minor
    xlabel('Samples')
    ylabel('|usb|, |lsb| in mV')

    p2 = subplot(212);
    plot(hf_sig * 1000)
    hold on
    plot(abs(bb_sig) * 1000)
    linkaxes([p1, p2], 'x')
    title('hf signal and |Complex baseband signal|')
    legend('hf signal', '|Complex baseband signal|', 'Location', ...
        'southoutside')
    grid minor
    xlabel('Samples')
    ylabel('|baseband-signal|, |hf-signal| in mV')

end


% ------------- Coarse start and end of response detection----------------
% Calculate the first difference of the amplitudes of USB and LSB and
% find local maxima and minima.
%
% The reference points for the correlator are determined by the first
% maximum and the last minimum and maximum of USB and LSB


sig = diff(abs(usb));
[usb_vals_max, usb_idx_max] = new_findpeaks(sig);
threshold = 0.5 * max(usb_vals_max);
threshold = mean(usb_vals_max(usb_vals_max > threshold));
usb_idx_max(usb_vals_max < 0.5 * threshold) = [];
usb_vals_max(usb_vals_max < 0.5 * threshold) = [];


sig = diff(abs(lsb));
[lsb_vals_max, lsb_idx_max] = new_findpeaks(sig);
threshold = 0.5 * max(lsb_vals_max);
threshold = mean(lsb_vals_max(lsb_vals_max > threshold));
lsb_idx_max(lsb_vals_max < 0.25 * threshold) = [];
lsb_vals_max(lsb_vals_max < 0.25 * threshold) = [];


%Minima of USB and LSB
sig = diff(abs(usb));
[usb_vals_min, usb_idx_min] = new_findpeaks(-sig);
threshold = 0.5 * max(usb_vals_min);
threshold = mean(usb_vals_min(usb_vals_min > threshold));
usb_idx_min(usb_vals_min < 0.25 * threshold) = [];
usb_vals_min(usb_vals_min < 0.25 * threshold) = [];
usb_vals_min = - usb_vals_min;


sig = diff(abs(lsb));
[lsb_vals_min, lsb_idx_min] = new_findpeaks(-sig);
```

```
threshold = 0.5 * max(lsb_vals_min);
threshold = mean(lsb_vals_min(lsb_vals_min > threshold));
lsb_idx_min(lsb_vals_min < 0.25 * threshold) = [];
lsb_vals_min(lsb_vals_min < 0.25 * threshold) = [];
lsb_vals_min = - lsb_vals_min;

%Correlator wavelet
wavelet = [ones(round(samples_per_subcarrier_cycle / 2), 1); ...
    -ones(round(samples_per_subcarrier_cycle / 2), 1)];

% ------------- Exact start and end of response detection ----------------

% ----- Exact start of response detection
scaler = .25;
detection_range_start = round(mean([usb_idx_max(1), lsb_idx_max(1)]) * ...
    fft_step);
detection_range = detection_range_start : detection_range_start + ...
    fft_length;

signal_energy_real = max(xcov(real(bb_sig(detection_range))));
signal_energy_imag = max(xcov(imag(bb_sig(detection_range))));

if signal_energy_real > signal_energy_imag
    sig = real(bb_sig(detection_range));
    if DEBUG
        fprintf(['Real part of baseband signal has higher signal ', ...
            'power at start of PICC response.\nUsing this part for ', ...
            'detecting the exact start.\n\r'])
    end
else
    sig = imag(bb_sig(detection_range));
    if DEBUG
        fprintf(['Imaginary part of baseband signal has higher ', ...
            'signal power at start of PICC response.\nUsing this ', ...
            'part for detecting the exact start.\n\r'])
    end
end


sig = [sig(1) * ones(length(wavelet), 1); sig; sig(end) * ...
    ones(length(wavelet), 1)];
N_sig = length(sig);
xc = xcov(sig, wavelet);
xc = xc(N_sig : end);


sig_ = xc;
[max_vals, max_idx] = new_findpeaks(sig_);
threshold = 0.5 * max(max_vals);
threshold = mean(max_vals(max_vals > threshold));
max_idx(max_vals < scaler * threshold) = [];
max_vals(max_vals < scaler * threshold) = [];


sig_ = xc;
[min_vals, min_idx] = new_findpeaks(-sig_);
threshold = 0.5 * max(min_vals);
threshold = mean(min_vals(min_vals > threshold));
min_idx(min_vals < scaler * threshold) = [];
min_vals(min_vals < scaler * threshold) = [];
min_vals = - min_vals;

index_correction = detection_range_start - length(wavelet) + ...
    round(samples_per_subcarrier_cycle / 2);

if min_idx(1) < max_idx(1)
    start_idx = min_idx(1) + index_correction;
else
    start_idx = max_idx(1) + index_correction;
end
```

```
% ----- Exact end of response detection
scaler = .5;
detection_range_start = round(mean([usb_idx_min(end), ...
    lsb_idx_min(end)]) * fft_step);

detection_range = detection_range_start : length(bb_sig);

signal_energy_real = max(xcov(real(bb_sig(detection_range))));
signal_energy_imag = max(xcov(imag(bb_sig(detection_range))));

if signal_energy_real > signal_energy_imag
    sig = real(bb_sig(detection_range));
    if DEBUG
        fprintf(['Real part of baseband signal has higher signal ', ...
            'power at end of PICC response.\nUsing this part for ', ...
            'detecting the exact end.\n\r'])
    end
else
    sig = imag(bb_sig(detection_range));
    if DEBUG
        fprintf(['Imaginary part of baseband signal has higher ', ...
            'signal power at end of PICC response.\nUsing this part ', ...
            'for detecting the exact end.\n\r'])
    end
end

sig = [sig(1) * ones(length(wavelet), 1); sig; sig(end) * ...
    ones(length(wavelet), 1)];
N_sig = length(sig);
xc = xcov(sig, wavelet);
xc = xc(N_sig : end - length(wavelet) - 1);

sig_ = xc;
[max_vals, max_idx] = new_findpeaks(sig_);
threshold = 0.5 * max(max_vals);
threshold = mean(max_vals(max_vals > threshold));
max_idx(max_vals < scaler * threshold) = [];
max_vals(max_vals < scaler * threshold) = [];

sig_ = xc;
[min_vals, min_idx] = new_findpeaks(-sig_);
threshold = 0.5 * max(min_vals);
threshold = mean(min_vals(min_vals > threshold));
min_idx(min_vals < scaler * threshold) = [];
min_vals(min_vals < scaler * threshold) = [];
min_vals = - min_vals;


%Offset for the determined index to correspond to the baseband signal
index_correction = detection_range_start - length(wavelet) + ...
    round(samples_per_subcarrier_cycle / 2);

% Evaluate the local extrema of the correlation signal.
% The end of response is found if two extrema if inverse type are found
% within 1.1 subcarriers.
end_found = 0;
extrema = [max_idx, ones(length(max_idx), 1);
           min_idx, zeros(length(min_idx), 1)];
extrema = sortrows(extrema);
% Swap the rows of the matrix so that the last extrema is located first
extrema = flipud(extrema);

for k = 1 : length(extrema) - 1

    if extrema(k + 1, 2) == ~extrema(k, 2) && ...
        abs(extrema(k, 1) - extrema(k + 1, 1)) < 1.1 * ...
            samples_per_subcarrier_cycle

        tmp_end_idx = extrema(k, 1);
```

```
        end_found = 1;
        break;

    end
end

if ~end_found
    warning(['Error. No valid end of response detected. ', ...
        'Terminating script.'])
    ret_val = -1;
    return
end
end_idx = tmp_end_idx + index_correction;


if PLOT
     fprintf(['Plot functionality in sliding_lma() enabled. This ', ...
         'will generate plots showing how the start and end of ', ...
         'the PICC response are detected.\n\r']);
    figure()
    fprintf(['\tFigures %dff show the real and imaginary part of ', ...
        'the complex baseband signal with the detected start and ', ...
        'end of the PICC response. \n\r'], ...
        get(gcf, 'Number'))
    p1 = subplot(211);
    plot(real(bb_sig))
    hold on
    plot(start_idx, real(bb_sig(start_idx)), 'co')
    plot(end_idx, real(bb_sig(end_idx)), 'c*')
    grid minor
    title(['Real(Baseband) signal with detected start and end ', ...
        'of response.'])
    xlabel('Samples')
    legend('Real(Baseband signal)', 'Detected start', 'Detected end')


    p2 = subplot(212);
    plot(imag(bb_sig))
    hold on
    plot(start_idx, imag(bb_sig(start_idx)), 'co')
    plot(end_idx, imag(bb_sig(end_idx)), 'c*')
    grid minor
    title(['Imag(Baseband) signal with detected start and end ', ...
        'of response.'])
    xlabel('Samples')
    legend('Imag(Baseband signal)', 'Detected start', 'Detected end')

    linkaxes([p1, p2], 'x')
end
end
```

N.12.6 subcarrier_analysis.m

```
function [subcarrier_states, bit_states, PD_struct, subcarrier_indices,...
    ignore_last_subcarrier, ret_val] = ...
    subcarrier_analysis(bb_sig_filt, start_idx, end_idx, param, ...
    cal_coil_signal)

% function [subcarrier_states, bit_states, PD_struct, subcarrier_indices,
%    ignore_last_subcarrier, ret_val] = ...
%    subcarrier_analysis(bb_sig_filt, start_idx, end_idx, param, ...
%    cal_coil_signal)
%
%
% DESCRIPTION: Function for determining the subcarrier states of the
%              PICC response. The inter and intra phase drift according
%              to ISO/IEC 10373-6:2017 Annex N are computed
%
% INPUT:
%        bb_sig_filt.....filtered complex baseband signal
```

```
%        start_idx.......Start index of the PICC response
%        end_idx.........End index of the PICC response
%        param...........Structure containing information about the signal
%                        (sampling frequency, signal encoding etc.)
%
% OUTPUT:
%       subcarrier_states.....determined subcarrier states of the
%                             signal:
%                             1: State MS1 in the first half, state MS2/US
%                                in the second half
%                             0: State MS2/US in the first half, state MS1
%                                in the second half
%                            -1: State could not be determined (Type B,
%                                Type A > 106 kBps), no subcarrier
%                                modulation (Type A, 106 kBps)
%       bit_states...........Only relevant for Type A 106kBps responses;
%                            Logic state of the corresponding bit
%                            (logic 0 or logic 1)
%       ret_val..............error flag: If not zero, an error occurred
%

% Parameter initialization
PD_struct = [];
ret_val = 0;

fsamp = param.fsamp;
fc = param.fc;
subcarrier_divider = param.f_sub_div;
encoding = param.encoding;

fsub = fc / subcarrier_divider;
samples_per_subcarrier_cycle = round(fsamp / fsub);

% setup initial subcarrier grid
subcarrier_indices = initial_subcarrier_grid(start_idx, ...
    end_idx, samples_per_subcarrier_cycle);


% --------- Subcarrier analysis for Type A, 106 kBps responses -----------
if strcmp(encoding.interface_type, 'Type_A') && ...
        strcmp(encoding.bitrate, '106')
    [bit_states, subcarrier_states, subcarrier_indices, ...
        ignore_last_subcarrier, ret_val] = ...
        get_subcarrier_states_Type_A_106(...
        bb_sig_filt, samples_per_subcarrier_cycle, ...
        subcarrier_indices);

    if ret_val ~= 0
        state_determination_error = 1;
    else
        state_determination_error = 0;
    end

    if ignore_last_subcarrier
        subcarrier_states = subcarrier_states(1 : end - 1);
        subcarrier_indices = subcarrier_indices(1 : end - 1);
    end


% ----------- Subcarrier analysis for all other responses ----------------
else
    [subcarrier_states, ~, ...
        subcarrier_indices, ignore_last_subcarrier, ...
        state_determination_error] = ...
    get_subcarrier_states_Type_B_A_not_106(bb_sig_filt, ...
    subcarrier_indices, encoding);

    % For Type B signals and Type A signals with a bitrate higher than
    % 106 kBps separate decoding functions must be called to get the bit
    % states.
    bit_states = -1;
```

```
    if state_determination_error == false && subcarrier_states(end) == -1
        subcarrier_indices = subcarrier_indices(1 : end - 1);
        subcarrier_states = subcarrier_states(1 : end - 1);
    end

end

if state_determination_error ~= 0
    ret_val = -1;
    return
end

% Tests showed that due to transient effects the last subcarrier in the
% response has to be excluded from phase drift computation
subcarrier_states_for_phase_drift_computation = subcarrier_states( ...
    1 : end - 1);
subcarrier_indices_for_phase_drift_computation = subcarrier_indices( ...
    1 : end - 1);


% Phase drift computation
PD_struct = phase_drift_computation( ...
    subcarrier_states_for_phase_drift_computation, ...
    subcarrier_indices_for_phase_drift_computation, ...
    samples_per_subcarrier_cycle, bb_sig_filt, encoding, param, ...
    cal_coil_signal);



end



function subcarrier_indices = initial_subcarrier_grid( ...
    start_idx, end_idx, samples_per_subcarrier_cycle)

% function subcarrier_indices = initial_subcarrier_grid( ...
%    SOC_idx, EOC_idx, samples_per_subcarrier_cycle)
%
% DESCRIPTION: generates the initial subcarrier grid for a ISO 14443
%              Type-A or Type-B PICC response
%
%              The subcarrier cycles are calculated from SOC every
%              samples_per_subcarrier_cycle
%
%
% INPUT:
%       start_idx.....................Start of the signal
%       end_idx.......................End of the signal
%       samples_per_subcarrier_cycle...Samples in one subcarrier cycle
%
% OUTPUT:
%       subcarrier_indices...Vector containing the subcarrier cycles
%

number_of_subcarrier_cycles = ceil((end_idx - start_idx) / ...
    samples_per_subcarrier_cycle);

subcarrier_indices = (0 : number_of_subcarrier_cycles) * ...
    samples_per_subcarrier_cycle + start_idx;

%Every index vector in the script is a row vector
subcarrier_indices = subcarrier_indices.';
end
```

N.12.7 phase_drift_computation.m

```
function PD_struct = phase_drift_computation(subcarrier_states, ...
    subcarrier_indices, samples_per_subcarrier_cycle, bb_sig_filt, ...
```

```
    encoding, param, cal_coil_signal)

% function PD_struct = phase_drift_computation(subcarrier_states, ...
%    subcarrier_indices, samples_per_subcarrier_cycle, bb_sig_filt, ...
%    encoding, param, cal_coil_signal)
%
% DESCRIPTION: Function for computing the Inter- and Intrastate phase
%              drift as specified in ISO/IEC 10373-6:2018 Annex N
%
% INPUT:
%       subcarrier_states ..............Subcarrier states
%       subcarrier_indices..............Subcarrier boundaries
%       samples_per_subcarrier_cycle.....Samples in one subcarrier cycle
%       bb_sig_filt.....................Filtered complex baseband signal
%       encoding .......................The encoding of the PICC response
%       param...........................Structure containing information
%                                       about the signal (sampling rate
%                                       etc.)
%       cal_coil_signal.................Signal from the calibration coil
% OUTPUT:
%
%       PD_struct...structure containing the following variables
%           .fig_delta_const...figure handle
%           .fig_phase....figure handle
%           .delta_pd_pp..peak-peak phase drift of delta signal [radians]
%           .min_mag_val..minimum magnitude value of state MS1 over time
%           .mag_limit....magnitude limit (requirement), = V_lma_min
%           .phi_US.......angle of unmodulated field
%           .h_field......field strength [A/m(rms)]
%           .phi_initial..angle of complex vector difference of first
%                         MS1 occurrence and subsequent MS2 occurrence
%
%
SAVE = 0;

[MS1_sig, MS2_sig, us_sig, trans, phi_initial, MS1_sig_m, ...
    MS2_sig_m] = ...
    extract_state_samples(...
    bb_sig_filt, subcarrier_states, subcarrier_indices, encoding, ...
    samples_per_subcarrier_cycle);

%---------- Initial state US amplitude and phase computation -------------
N_carrier = 64; % number of unmodulated carrier periods;
samples_per_carrier = round(param.fsamp / param.fc);
Sil_len_samp = 1 : (N_carrier * samples_per_carrier);
backoff = 2*samples_per_carrier;
Sil_len_samp = max(1, subcarrier_indices(1) - Sil_len_samp - backoff);

mu_sil = mean(bb_sig_filt( Sil_len_samp ));

phi_US = angle(mu_sil) * 180 / pi; % initial phase [degrees]
loading_effect_US = abs(mu_sil);
fprintf('Initial PICC carrier amplitude (US): %2.2f V\r\n', ...
    loading_effect_US)
fprintf('Initial PICC carrier phase (US): %2.2f degrees\r\n', phi_US)




%---- Compute Vlma limit as a function of PICC class and field strength---

V_rms_cal_coil = rms(cal_coil_signal(1:round(50*(param.fsamp / ...
    param. fc))));
[V_lma_min, V_lma_max, h_field] = get_Vlma_limit(V_rms_cal_coil, ...
    param.PICC_class);


%--------------- Compute amplitude and phase drift-----------------------

[PD_struct] = compute_phase_drift(MS1_sig, MS2_sig, mu_sil, V_lma_min, ...
    length(subcarrier_states));
```

```
%------Compute max PD within each state occurrence and max phase drift
% across all state occurrences-----------------------------------------

[PD_struct.max_intraState_PD, PD_struct.max_interState_PD, ...
    PD_struct.h_inter_intraPD] =...
    phase_drift_within_and_over_state(MS1_sig_m, MS2_sig_m);

PD_struct.phi_US = phi_US; % angle of unmodulated PICC field
PD_struct.h_field = h_field;
PD_struct.V_lma_max = V_lma_max;
PD_struct.phi_initial = phi_initial; % angle of complex vector difference
                                     % of first MS1 occurrence and
                                     % subsequent MS2 occurrence

%--------------------- Plot constellation diagram-------------------------
h_fig_td_cBB = plot_time_continous_const_plot(...
    us_sig, trans, MS1_sig, MS2_sig);

if SAVE
    store_name = param.name;
   if isfield(PD_struct, 'fig_phase')
       saveas(PD_struct.fig_phase, [store_name, '_MS1MS2_state'], 'fig');
       saveas(PD_struct.fig_phase, [store_name, '_MS1MS2_state'], 'png');
   end
   if exist('h_fig_td_cBB')
       saveas(h_fig_td_cBB, [store_name, '_constDia_trans'], 'fig');
       saveas(h_fig_td_cBB, [store_name, '_constDia_trans'], 'png');
   end
   if isfield(PD_struct, 'h_inter_intraPD')
       saveas(PD_struct.h_inter_intraPD, ...
           [store_name, '_inter_intraPD'], 'fig');
       saveas(PD_struct.h_inter_intraPD, ...
           [store_name, '_inter_intraPD'], 'png');
   end
end
end
```

N.12.8 signal_decoding.m

```
function [bit_states, SOF_start, ret_val] = signal_decoding(...
    subcarrier_states, encoding, Type_A_ignore_last_subcarrier)

% function [bit_states, SOF_start, ret_val] = signal_decoding(...
%    subcarrier_states, encoding, Type_A_ignore_last_subcarrier)
%
% DESCRIPTION: Function for the interface-type and bitrate dependent
%              decoding of the PICC response
%
% INPUT:
%       subcarrier_states.....Vector containing the subcarrier states
%       encoding.....The encoding of the signal
%       Type_A_ignore_last_subcarrier.....Flag, 1b to indicate that last
%           subcarrier should be ignored
%
% OUTPUT:
%       bit_states.....Matrix containing the subcarrier states of each
%       SOF_start.....SOF index
%       ret_val.....error flag: If not zero, an error warning was
%                       triggered during signal decoding

ret_val = 0;
SOF_start = 0;
bit_states = -1;
if strcmp(encoding.interface_type, 'Type_B')
    [bit_states_in_characters, SOF_start, ret_val] = ...
        Type_B_decoding(subcarrier_states, encoding);

    if ret_val ~= 0
        return
```

```
    end

    % Transform bit state matrix to vector
    bit_states = bit_states_in_characters(:, 2 : 9);
    bit_states = bit_states';
    bit_states = bit_states(:);

elseif strcmp(encoding.interface_type, 'Type_A') && ...
        ~strcmp(encoding.bitrate, '106')

    [bit_states, ret_val] = ...
        Type_A_not_106_decoding(subcarrier_states, encoding, ...
        Type_A_ignore_last_subcarrier);

else
    bit_states = [];
end

end
```

N.12.9 compute_phase_drift.m

```
function [PD_struct] = compute_phase_drift(MS1_sig, MS2_sig, mu_sil, ...
    V_lma_min, N_subcarrier)

% function [PD_struct] = compute_phase_drift(MS1_sig, MS2_sig, mu_sil, ...
%    V_lma_min, N_subcarrier)
%
% DESCRIPTION: computes the phase drift as well as the amplitude
% requirements as defined in ISO/IEC 14443, compares the measured values
% to the limit values and visualizes the results.
%
% INPUT:
%       MS1_sig.....Samples of signal belonging to state MS1 occurrences
%       MS1_idx.....index vector indicating the time positions of MS1
%                   occurrences within signal
%       N_subcarrier.....Number of subcarriers in the PICC response
% OUTPUT:
%       PD_struct.....structure containing the following variables:
%           PD_struct.fig_phase.....Figure handle
%           PD_struct.min_mag_val.....Minimum magnitude value of state MS1
%               over time
%           PD_struct.V_lma_min.....Magnitude limit (requirement),
%               identical to V_lma_min

lw = 2; % plot: line width
h_fig = figure('color', [1 1 1]);
scrsz = get(0, 'ScreenSize');
set(h_fig, 'position',...
    [scrsz(3) / 2, scrsz(4) * .47, scrsz(4) * .6, scrsz(4) * .4])

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% MS1 and MS2 magnitude computation (related to US) and visualization

amp_MS1 = abs(MS1_sig - mu_sil) * 1e3; % mV
amp_MS2 = abs(MS2_sig - mu_sil) * 1e3; % mV

% compute number of subcarrier
N = max(length(MS1_sig), length(MS2_sig));
%N_subcarrier = length(MS1_idx);
samp_per_subcarrier = N / N_subcarrier;

t = 1 : 1 / samp_per_subcarrier : N_subcarrier + 1;
t = t(1 : N);

figure(h_fig),

plot(t, amp_MS1, 'k', 'linewidth', lw)
hold on
plot(t(1 : length(MS2_sig)), (amp_MS2), 'b', 'linewidth', lw)
```

```
plot(t, ones(size(t)) * V_lma_min, 'r--',  'linewidth', lw)

title('MS1-US and MS2-US modulus over time')
grid on
xlabel('Subcarrier periods')
ylabel('Modulus (mV)')
legend('MS1-US modulus', 'MS2-US modulus', 'Modulus limit')
axis tight


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% informative Parameters
PD_struct.fig_phase = h_fig;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% normative Parameters
PD_struct.min_mag_val = min(amp_MS1); % min amplitude of MS1 related to US
% state
PD_struct.V_lma_min = V_lma_min; % threshold for the min amplitude of MS1
```

N.12.10 correct_f_error.m

```
function [f_err] = correct_f_error(in, strt_idx, stop_idx, fsamp, fc)

% function [f_err] = correct_f_error(in, strt_idx, stop_idx, fsamp, fc)
%
% DESCRIPTION: performs an estimation of the frequency error resulting
%              in a constant phase drift ==> estimates the slope
%
% INPUT:
%       in........complex IQ input signal
%       strt_idx..analysis start index [samples]
%       stop_idx..analysis stop index [samples]
%       fsamp.....sampling frequency [Hz]
%       fc........carrier frequency [Hz]
% OUTPUT:
%       f_err.....estimated frequency error

in = in(:);

% perform IIR filtering to suppress high frequency components
% (cut-off frequency is set to 1 MHz)
[b,a] = butter(4, 1e6 / (fsamp / 2));
x = filter(b, a, real(in)) + 1i * filter(b, a, imag(in)); % filter signal
% remove filter fade-in transient from signal
in = x(max(length(b), round(30 * fsamp / fc) + 1 : end));
stop_idx = min(stop_idx, length(in));

in = in(strt_idx : stop_idx);
N = length(in);
t = (0 : 1 / fsamp : (N - 1) / fsamp)';

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

f_norm = EstimateCarrierFreq(in);
f_err = f_norm * fsamp;
if isnan(f_err)
    f_err = 0;
end


function FcEst = EstimateCarrierFreq(in_bb_unmod)

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

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

in_bb_unmod = in_bb_unmod(:);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 1) Rough phase drift estimation
in_bb_unmod1 = in_bb_unmod ./ abs(in_bb_unmod); % complex absolute value

% Totally remove amplitude fluctuations
IQJumps = diff(in_bb_unmod1); % phasor speed (instantaneous)

% Average of abs of differentiated phasor signal
IQjumpsMeanAbs = mean(abs(IQJumps));
FcEst = asin(IQjumpsMeanAbs / 2) / pi;

% Frequency rotation
N1 = length(in_bb_unmod);
derotatedPhasor = in_bb_unmod.*exp(-1i * 2 * pi * FcEst * (1 : N1).');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 2) Fine phase drift estimation

phaseSignalStep2 = unwrap(angle(derotatedPhasor));

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;
```

N.12.11 extract_state_samples.m

```
function [MS1_sig, MS2_sig, us_sig, trans, phi_initial, MS1_sig_m, ...
    MS2_sig_m] = ...
    extract_state_samples(...
    bb_sig_filt, subcarrier_states, subcarrier_indices, encoding, ...
    samples_per_subcarrier_cycle)

% function [MS1_sig, MS2_sig, us_sig, trans, phi_initial, MS1_sig_m, ...
%    MS2_sig_m] = ...
%    extract_state_samples(...
%    bb_sig_filt, subcarrier_states, subcarrier_indices, encoding, ...
%    samples_per_subcarrier_cycle)
%
% DESCRIPTION: Function for extracting the samples of the complex baseband
%              signal corresponding to MS1, MS2, US and the transitions
%              between the states for a Type B signal or a Type A signal
%              with a bitrate higher than 106 kBps
%
%              The function also calculates the initial phase of the
%              signal.
%
%
% INPUT:
%       bb_sig_filt.....The complex baseband signal
%       subcarrier_states.....Vector containing the subcarrier states
%       subcarrier_indices.....Vector containing the indices of  the
%                              subcarrier boundaries
%       encoding.....The encoding of the signal
%       samples_per_subcarrier_cycle.....Samples in one subcarrier
%
% OUTPUT:
%       MS1_sig.....Vector containing the samples corresponding to MS1
%       MS2_sig.....Vector containing the samples corresponding to MS2
%       us_sig.....Vector containing the samples corresponding to US
```

```
%       trans.....Vector containing the samples of the state transitions
%       phi_initial.....Initial phase of the signal
%       MS1_sig_m.....Matrix containing the samples of each occurrence
%                     of MS1 as columns
%       MS2_sig_m.....Matrix containing the samples of each occurrence
%                     of MS2 as columns

PLOT = 0;
encoding_parameters = get_encoding_parameters(encoding);
samples_per_carrier = samples_per_subcarrier_cycle / ...
    encoding_parameters.subcarrier_divider;

if strcmp(encoding.interface_type, 'Type_A') && ...
        strcmp(encoding.bitrate, '106')
    Type_A_106 = 1;
else
    Type_A_106 = 0;
end

% Threshold boundary specified how much samples are excluded from the
% phase drift computation starting at the subcarrier boundaries
threshold_boundary_start = round(0.17 * samples_per_subcarrier_cycle);
threshold_boundary_end = round(0.08 * samples_per_subcarrier_cycle);
state_occurrence_length = round(0.20 * samples_per_subcarrier_cycle);


if ~Type_A_106

    [MS1_sig, MS2_sig, trans, MS1_sig_m, MS2_sig_m, MS1_idx, MS2_idx, ...
        trans_idx] = ...
        state_sample_extreaction_Type_B_A_not_106(...
        bb_sig_filt, subcarrier_states, subcarrier_indices, ...
        samples_per_subcarrier_cycle, threshold_boundary_start, ...
        threshold_boundary_end, state_occurrence_length);
else
    [MS1_sig, MS2_sig, trans, MS1_sig_m, MS2_sig_m, MS1_idx, MS2_idx, ...
        trans_idx] = ...
        state_sample_extreaction_Type_A_106(...
        bb_sig_filt, subcarrier_states, subcarrier_indices, ...
        samples_per_subcarrier_cycle, threshold_boundary_start, ...
        threshold_boundary_end, state_occurrence_length);
end

%Extraction of samples of state US
us_state_end = subcarrier_indices(1) - samples_per_carrier * 2;
us_state_length = 64 * samples_per_carrier;

us_state_range = max(1, us_state_end - us_state_length + 1) : ...
    us_state_end;
us_sig = bb_sig_filt(us_state_range);



%Calculation of the initial phase
MS1_state1 = MS1_sig_m(1,:);
MS2_state1 = MS2_sig_m(1, :);
phi_initial = mean(angle(MS1_state1 - MS2_state1));

if PLOT

figure()

fprintf(['Plot functionality in extract_state_samples() enabled. ', ...
        '\n\tFigure %d shows the real and imaginary parts of the ', ...
        'extracted state samples.\n\r'], ...
        get(gcf, 'Number'))

p1 = subplot(211);
l1 = plot(MS1_idx, real(MS1_sig_m.'), 'b');
hold on
l2 = plot(MS2_idx, real(MS2_sig_m.'), 'g');
```

```
l3 = plot(trans_idx, real(bb_sig_filt(trans_idx)), 'Marker', 'o', ...
    'MarkerSize', 1, 'LineStyle', 'none', 'MarkerEdgeColor', ...
    [0.9, 0.9, 0.9]);
l4 = plot(us_state_range, real(us_sig), 'Color', [0,0,0]);
title('Real part of the extracted state samples')
legend([l1(end), l2(end), l3(end), l4],{'MS1', 'MS2', 'trans', 'US'})
grid minor

p2 = subplot(212);
l1 = plot(MS1_idx, imag(MS1_sig_m.'), 'b');
hold on
l2 = plot(MS2_idx, imag(MS2_sig_m.'), 'g');
l3 = plot(trans_idx, imag(bb_sig_filt(trans_idx)), 'Marker', 'o', ...
    'MarkerSize', 1, 'LineStyle', 'none', 'MarkerEdgeColor', ...
    [0.9, 0.9, 0.9]);
l4 = plot(us_state_range, imag(us_sig), 'Color', [0,0,0]);
title('Imaginary part of the extracted state samples')
legend([l1(end), l2(end), l3(end), l4],{'MS1', 'MS2', 'trans', 'US'})
grid minor
linkaxes([p1, p2], 'x')

end
end




function [MS1_sig, MS2_sig, trans, MS1_sig_m, MS2_sig_m, MS1_idx, ...
    MS2_idx, trans_idx] = ...
    state_sample_extreaction_Type_B_A_not_106(...
    bb_sig_filt, subcarrier_states, subcarrier_indices, ...
    samples_per_subcarrier_cycle, threshold_boundary_start, ...
    threshold_boundary_end, state_occurrence_length)

% function [MS1_sig, MS2_sig, trans, MS1_sig_m, MS2_sig_m, MS1_idx, ...
%    MS2_idx, trans_idx] = ...
%    state_sample_extreaction_Type_B_A_not_106(...
%    bb_sig_filt, subcarrier_states, subcarrier_indices, ...
%    samples_per_subcarrier_cycle, threshold_boundary_start, ...
%    threshold_boundary_end, state_occurrence_length)
%
% DESCRIPTION: Function for extracting the samples of the complex baseband
%              signal corresponding to MS1, MS2, US and the transitions
%              between the states
%
%              The function also calculates the initial phase of the
%              signal.
%
%
% INPUT:
%       bb_sig_filt.....The complex baseband signal
%       subcarrier_states.....Vector containing the subcarrier states
%       subcarrier_indices.....Vector containing the indices of  the
%                              subcarrier boundaries
%       encoding.....The encoding of the signal
%       samples_per_subcarrier_cycle.....Samples in one subcarrier
%       threshold_boundary_start.....Value specifying how much samples
%                                    are excluded from the start of the
%                                    subcarrier
%       threshold_boundary_end.....Value specifying how much samples are
%                                  excluded from the end of the subcarrier
%       state_occurrence_length.....The length of one state occurrence
%
% OUTPUT:
%       MS1_sig.....Vector containing the samples corresponding to MS1
%       MS2_sig.....Vector containing the samples corresponding to MS2
%       trans.....Vector containing the samples of the state transitions
%       MS1_sig_m.....Matrix containing the samples of each occurrence of
%                     MS1 as columns
%       MS2_sig_m.....Matrix containing the samples of each occurrence of
```

```
%                     MS2 as columns

% Initial subcarrier state with state 0 denotes a shifted subcarrier grid
% for a Type B signal. Tests showed that in this case the last subcarrier
% has to be excluded from phase drift calculation due to none-ideal
% behaviour which might distort the measurement results
if subcarrier_states(1) == 0
    subcarrier_states = subcarrier_states(1 : end - 1);
end

MS1_idx = -ones(length(subcarrier_states), state_occurrence_length);
MS2_idx = -ones(length(subcarrier_states), state_occurrence_length);
trans_idx = -ones(samples_per_subcarrier_cycle - 2 * ...
    state_occurrence_length, length(subcarrier_states));


% When tmp1 and tmp2 are XOR'd, the resulting vector will contain a '1'
% when the subcarrier state changed from the last to the current
% subcarrier cycle.
tmp1 = [subcarrier_states; subcarrier_states(end)];
tmp2 = [subcarrier_states(1); subcarrier_states];

phase_jumps = xor(tmp1, tmp2);
phase_jump_idx = find(phase_jumps == 1);
phase_jump_idx = (phase_jump_idx(:))';


% Depending on the last subcarrier the different samples have to be
% excluded from the phase drift computation. Seven different states have
% to be considered:
% State 0: MS2 first, MS1 second; States excluded from left and right
%          boundaries and from center
% State 1: MS1 first, MS2 second; States excluded from left and right
%          boundaries and from center
% State 2: MS2 first, MS1 second; Phase jump at end of subcarrier, so no
%          samples are excluded at end of subcarrier
% State 3: MS1 first, MS2 second; Phase jump at start of subcarrier, so no
%          samples are excluded at start of subcarrier
% State 4: MS1 first, MS2 second; Phase jump at end of subcarrier, so no
%          samples are excluded at end of subcarrier
% State 5: MS2 first, MS1 second; Phase jump at start of subcarrier, so no
%          samples are excluded at start of subcarrier
% State 6: MS1 first, MS2 second; Phase jump at both subcarrier
%          boundaries, so no samples are excluded at the subcarrier
%          boundaries
% State 7: MS2 first, MS1 second; Phase jump at both subcarrier
%          boundaries, so no samples are excluded at the subcarrier
%          boundaries

for k = phase_jump_idx
    if ismember(subcarrier_states(k-1), [0,1])
        if subcarrier_states(k - 1) == 1 && subcarrier_states(k) == 0
            subcarrier_states(k - 1) = 4;
            subcarrier_states(k) = 5;
        else
            subcarrier_states(k - 1) = 2;
            subcarrier_states(k) = 3;
        end
    else
        if subcarrier_states(k - 1) == 3
            subcarrier_states(k - 1) = 6;
            subcarrier_states(k) = 5;
        elseif subcarrier_states(k - 1) == 5
            subcarrier_states(k - 1) = 7;
            subcarrier_states(k) = 3;
        end
    end
end


%Calculate indices samples to extract depending on the subcarrier state
```

```
for k = 1 : length(subcarrier_states)
    current_state = subcarrier_states(k);

    if current_state == 0
        MS1_idx_start = subcarrier_indices(k + 1) - ...
            threshold_boundary_end - state_occurrence_length + 1;
        MS1_idx_end = subcarrier_indices(k + 1) - threshold_boundary_end;

        MS2_idx_start = subcarrier_indices(k) + threshold_boundary_start;
        MS2_idx_end = subcarrier_indices(k) + ...
            threshold_boundary_start + state_occurrence_length - 1;

        MS1_idx(k, :) = MS1_idx_start : MS1_idx_end;
        MS2_idx(k, :) = MS2_idx_start : MS2_idx_end;

        idx = [...
            (subcarrier_indices(k) : MS2_idx_start - 1)';
            (MS2_idx_end + 1 : MS1_idx_start - 1)';
            (MS1_idx_end + 1 : subcarrier_indices(k + 1) - 1)'
            ];

        trans_idx(1 : length(idx), k) = idx;

    elseif current_state == 1
        MS1_idx_start = subcarrier_indices(k) + threshold_boundary_start;
        MS1_idx_end = subcarrier_indices(k) + ...
            threshold_boundary_start + state_occurrence_length - 1;

        MS2_idx_start = subcarrier_indices(k + 1) - ...
            threshold_boundary_end - state_occurrence_length + 1;
        MS2_idx_end = subcarrier_indices(k + 1) - threshold_boundary_end;

        MS1_idx(k, :) = MS1_idx_start : MS1_idx_end;
        MS2_idx(k, :) = MS2_idx_start : MS2_idx_end;

        idx = [...
            (subcarrier_indices(k) : MS1_idx_start - 1)';
            (MS1_idx_end + 1 : MS2_idx_start - 1)';
            (MS2_idx_end + 1 : subcarrier_indices(k + 1) - 1)'];

        trans_idx(1 : length(idx), k) = idx;

    elseif current_state == 2
        MS1_idx_start = subcarrier_indices(k + 1) - ...
            state_occurrence_length;
        MS1_idx_end = subcarrier_indices(k + 1) - 1;

        MS2_idx_start = subcarrier_indices(k) + threshold_boundary_start;
        MS2_idx_end = subcarrier_indices(k) + ...
            threshold_boundary_start + state_occurrence_length - 1;

        MS1_idx(k, :) = MS1_idx_start : MS1_idx_end;
        MS2_idx(k, :) = MS2_idx_start : MS2_idx_end;

        idx = [...
            (subcarrier_indices(k) : MS2_idx_start - 1)';
            (MS2_idx_end + 1 : MS1_idx_start - 1)'
            ];

        trans_idx(1 : length(idx), k) = idx;

    elseif current_state == 3
        MS1_idx_start = subcarrier_indices(k);
        MS1_idx_end = subcarrier_indices(k) + state_occurrence_length - 1;

        MS2_idx_start = subcarrier_indices(k + 1) - ...
            threshold_boundary_end - state_occurrence_length + 1;
        MS2_idx_end = subcarrier_indices(k + 1) - ...
            threshold_boundary_end;

        MS1_idx(k, :) = MS1_idx_start : MS1_idx_end;
```

```
        MS2_idx(k, :) = MS2_idx_start : MS2_idx_end;

        idx = [
            (MS1_idx_end + 1 : MS2_idx_start - 1)';
            (MS2_idx_end + 1 : subcarrier_indices(k + 1) - 1)'
            ];

        trans_idx(1 : length(idx), k) = idx;

    elseif current_state == 4
        MS1_idx_start = subcarrier_indices(k) + threshold_boundary_start;
        MS1_idx_end = subcarrier_indices(k) + ...
            threshold_boundary_start + state_occurrence_length - 1;

        MS2_idx_start = subcarrier_indices(k + 1) - ...
            state_occurrence_length;
        MS2_idx_end = subcarrier_indices(k + 1) - 1;

        MS1_idx(k, :) = MS1_idx_start : MS1_idx_end;
        MS2_idx(k, :) = MS2_idx_start : MS2_idx_end;

        idx = [...
            (subcarrier_indices(k) : MS1_idx_start - 1)';
            (MS1_idx_end + 1 : MS2_idx_start - 1)'
            ];

        trans_idx(1 : length(idx), k) = idx;

    elseif current_state == 5
        MS1_idx_start = subcarrier_indices(k + 1) - ...
            threshold_boundary_end - state_occurrence_length + 1;
        MS1_idx_end = subcarrier_indices(k + 1) - ...
            threshold_boundary_end;

        MS2_idx_start = subcarrier_indices(k);
        MS2_idx_end = subcarrier_indices(k) + state_occurrence_length - 1;

        MS1_idx(k, :) = MS1_idx_start : MS1_idx_end;
        MS2_idx(k, :) = MS2_idx_start : MS2_idx_end;

        idx = [...
            (MS2_idx_end + 1 : MS1_idx_start - 1)';
            (MS1_idx_end + 1 : subcarrier_indices(k+1) - 1)'
            ];

        trans_idx(1 : length(idx), k) = idx;

    elseif current_state == 6
        MS1_idx_start = subcarrier_indices(k);
        MS1_idx_end = subcarrier_indices(k) + state_occurrence_length - 1;

        MS2_idx_start = subcarrier_indices(k + 1) - ...
            state_occurrence_length;
        MS2_idx_end = subcarrier_indices(k + 1) - 1;

        MS1_idx(k, :) = MS1_idx_start : MS1_idx_end;
        MS2_idx(k, :) = MS2_idx_start : MS2_idx_end;

        idx = (MS1_idx_end + 1 : MS2_idx_start - 1)';
        trans_idx(1 : length(idx), k) = idx;

    elseif current_state == 7
        MS1_idx_start = subcarrier_indices(k + 1) - ...
            state_occurrence_length;
        MS1_idx_end = subcarrier_indices(k + 1) - 1;

        MS2_idx_start = subcarrier_indices(k);
        MS2_idx_end = subcarrier_indices(k) + state_occurrence_length - 1;

        MS1_idx(k, :) = MS1_idx_start : MS1_idx_end;
        MS2_idx(k, :) = MS2_idx_start : MS2_idx_end;
```

```
        idx = (MS2_idx_end + 1 : MS1_idx_start - 1)';
        trans_idx(1 : length(idx), k) = idx;
    end
end


%Extract samples
MS1_sig_m = bb_sig_filt(MS1_idx);
MS2_sig_m = bb_sig_filt(MS2_idx);

MS1_idx = MS1_idx';
MS2_idx = MS2_idx';

MS1_sig = MS1_sig_m.';
MS1_sig = MS1_sig(:);

MS2_sig = MS2_sig_m.';
MS2_sig = MS2_sig(:);

trans = bb_sig_filt(trans_idx(:));
end


function [MS1_sig, MS2_sig, trans, MS1_sig_m, MS2_sig_m, MS1_idx, ...
    MS2_idx, trans_idx] = ...
    state_sample_extreaction_Type_A_106(...
    bb_sig_filt, subcarrier_states, subcarrier_indices, ...
    samples_per_subcarrier_cycle, threshold_boundary_start, ...
    threshold_boundary_end, state_occurrence_length)

% function [MS1_sig, MS2_sig, trans, MS1_sig_m, MS2_sig_m, MS1_idx, ...
%    MS2_idx, trans_idx] = ...
%    state_sample_extreaction_Type_A_106(...
%    bb_sig_filt, subcarrier_states, subcarrier_indices, ...
%    samples_per_subcarrier_cycle, threshold_boundary_start, ...
%    threshold_boundary_end, state_occurrence_length)%

%
% DESCRIPTION: Function for extracting the samples of the complex baseband
%              signal corresponding to MS1, MS2, US and the transitions
%              between the states for a Type A signal with a bitrate of
%              106 kBps
%
%              The function also calculates the initial phase of the
%              signal.
%
%              For Type A, 106kBps signal the passed subcarrier states are
%              either 1 or -1. A subcarrier state of 1 occurs when the
%              corresponding half etu is subcarrier modulated. Subcarrier
%              states of -1 denotes no subcarrier modulation.
%
%
% INPUT:
%       bb_sig_filt.....The complex baseband signal
%       subcarrier_states.....Vector containing the subcarrier states
%       subcarrier_indices.....Vector containing the indices of the
%                              subcarrier boundaries
%       encoding.....The encoding of the signal
%       samples_per_subcarrier_cycle.....Samples in one subcarrier
%       threshold_boundary_start.....Value specifying how much samples are
%                                    excluded from the start of the
%                                    subcarrier
%       threshold_boundary_end.....Value specifying how much samples are
%                                  excluded from the end of the subcarrier
%       state_occurrence_length.....The length of one state occurrence
%
% OUTPUT:
%       MS1_sig.....Vector containing the samples corresponding to MS1
%       MS2_sig.....Vector containing the samples corresponding to MS2
%       trans.....Vector containing the samples of the state transitions
```

```
%       MS1_sig_m.....Matrix containing the samples of each occurrence of
%                     MS1 as columns
%       MS2_sig_m.....Matrix containing the samples of each occurrence of
%                     MS2 as columns


%Index matrices for MS1, MS2 and state transitions
MS1_idx = -ones(length(subcarrier_states), state_occurrence_length);
MS2_idx = -ones(length(subcarrier_states), state_occurrence_length);
trans_idx = -ones(samples_per_subcarrier_cycle, ...
    length(subcarrier_states));

for k = 1 : length(subcarrier_states)

    current_state = subcarrier_states(k);

    % If a subcarrier state of 1 occurs samples from the subcarrier
    % boundaries and
    % the subcarrier centers are excluded from phase drift computation and
    % stored to the vector of state transitions
    if current_state == 1
        MS1_idx_start = subcarrier_indices(k) + threshold_boundary_start;
        MS1_idx_end = subcarrier_indices(k) + ...
            threshold_boundary_start + state_occurrence_length - 1;

        MS2_idx_start = subcarrier_indices(k + 1) - ...
            threshold_boundary_end - state_occurrence_length + 1;
        MS2_idx_end = subcarrier_indices(k + 1) - threshold_boundary_end;

        MS1_idx(k, :) = MS1_idx_start : MS1_idx_end;
        MS2_idx(k, :) = MS2_idx_start : MS2_idx_end;

        idx = [...
            (subcarrier_indices(k) : MS1_idx_start - 1)';
            (MS1_idx_end + 1 : MS2_idx_start - 1)';
            (MS2_idx_end + 1 : subcarrier_indices(k + 1) - 1)'];

        trans_idx(1 : length(idx), k) = idx;

        % In case of a subcarrier state of -1 the whole samples of the
        % corresponding subcarrier are stored in the state transition
        % vector
    else
        idx = subcarrier_indices(k) : subcarrier_indices(k + 1) - 1;
        trans_idx(1 : length(idx), k) = idx;
    end

end

%Remove the excess matrix entries
MS1_idx(MS1_idx(:, 1) == -1, :) = [];
MS2_idx(MS2_idx(:, 1) == -1, :) = [];

trans_idx = trans_idx(:);
trans_idx(trans_idx == -1) = [];

%Extract samples

MS1_sig_m = bb_sig_filt(MS1_idx);
MS2_sig_m = bb_sig_filt(MS2_idx);

MS1_idx = MS1_idx';
MS2_idx = MS2_idx';

MS1_sig = MS1_sig_m.';
MS1_sig = MS1_sig(:);

MS2_sig = MS2_sig_m.';
MS2_sig = MS2_sig(:);

trans = bb_sig_filt(trans_idx(:));
```

```
end
```

N.12.12 evaluate_signal_power.m

```
function channel_selector = evaluate_signal_power(...
    modified_baseband_signal, subcarrier_indices)

%function channel_selector = evaluate_signal_power(...
%    modified_baseband_signal, subcarrier_indices)
%
% DESCRIPTION: Function for determining which part of the baseband signal
%              has to be ignored due to low signal power
%
% INPUT:
%       modified_baseband_signal.....Baseband signal with backoff
%       subcarrier_indices.....Corresponding subcarrier boundaried
%
%
%
%
% OUTPUT:
%      channel_selector.....'BOTH' : Use bith parts, 'REAL' : Use real
%                                 part,
%                           'IMAG' : Use imaginary part

global DEBUG;


% If the signal power of the real part is much smaller the power of the
% imaginary part, ignore the real part and vice versa
sig = modified_baseband_signal(subcarrier_indices(1) : ...
    subcarrier_indices(end));
sig = sig - mean(sig);

signal_power_real = rms(real(sig))^2;
signal_power_imag = rms(imag(sig))^2;
if DEBUG
    fprintf(['PICC-response signal power evaluation:\n'])
    fprintf(['\tSignal power of the real part of the whole PICC ', ...
        'response is %.2g mV^2\n\tSignal power of the imaginary ', ...
        'part of the whole PICC response is %.2g mV^2\n\r'], ...
    signal_power_real * 1000, signal_power_imag * 1000)
end

%Factor specifying threshold for determining the channel to use.
signal_power_factor = 2.5;

if signal_power_real < signal_power_imag / signal_power_factor
    channel_selector = 'IMAG';
    if DEBUG
        fprintf(['\tThe signal power of the imaginary part is more ', ...
            'then %3.1f times higher than the signal power of the ', ...
            'real part. \n\tTherefore only the imaginary part is ', ...
            'used for further calculations.\n\r'], ...
            signal_power_factor)
    end
elseif signal_power_imag < signal_power_real / signal_power_factor
    channel_selector = 'REAL';
    if DEBUG
        fprintf(['\tThe signal power of the real part is more then ', ...
            '%3.1f times higher than the signal power of the ', ...
            'imaginary part. \n\t', ...
            'Therefore only the real part is used for further ', ...
            'calculations.\n\r'], ...
        signal_power_factor)
    end
else
    channel_selector = 'BOTH';
    if DEBUG
        fprintf(['\tReal part and imaginary part of the PICC ', ...
```

```
            'response have a similar signal power. \n\tTherefore ', ...
            'using both signals for further calculations.\n\r'])
    end
end

end
```

N.12.13 get_encoding_parameters.m

```
function params = get_encoding_parameters(PICC_signal_encoding)
% function params = get_encoding_parameters(PICC_signal_encoding)
%
% DESCRIPTION: Function for acquiring some interface specific information.
%              e.g. subcarrier cycles in one bit duration (etu)
% INPUT:
%       PICC_signal_encoding...Encoding of the signal. Must be a structure
%       containing the following elements:
%           interface_type.....'Type_A' or 'Type_B'
%           bitrate.....'106', '212', '424', '848', '1695', '3390', ...
%                           or '6780'
% OUTPUT:
%       params.subcarrier_cycles_per_etu...Number of subarrier cycles in
%       one bit duration (etu)


interface_type = PICC_signal_encoding.interface_type;
bitrate = PICC_signal_encoding.bitrate;

params = struct();

switch interface_type
    case 'Type_A'
        switch bitrate
            case '106'
                params.subcarrier_cycles_per_etu = 8;
                params.subcarrier_divider = 16;
            case '212'
                params.subcarrier_cycles_per_etu = 4;
                params.subcarrier_divider = 16;
            case '424'
                params.subcarrier_cycles_per_etu = 2;
                params.subcarrier_divider = 16;
            case '848'
                params.subcarrier_cycles_per_etu = 1;
                params.subcarrier_divider = 16;
            case '1695'
                params.subcarrier_cycles_per_etu = 1;
                params.subcarrier_divider = 8;
            case '3390'
                params.subcarrier_cycles_per_etu = 1;
                params.subcarrier_divider = 4;
            case '6780'
                params.subcarrier_cycles_per_etu = 1;
                params.subcarrier_divider = 2;
            otherwise
               error('Unspecified bit rate in PICC_signal_encoding');
        end
    case 'Type_B'
         switch bitrate
            case '106'
                params.subcarrier_cycles_per_etu = 8;
                params.subcarrier_divider = 16;
            case '212'
                params.subcarrier_cycles_per_etu = 4;
                params.subcarrier_divider = 16;
            case '424'
                params.subcarrier_cycles_per_etu = 2;
                params.subcarrier_divider = 16;
            case '848'
                params.subcarrier_cycles_per_etu = 1;
```

```
                params.subcarrier_divider = 16;
            case '1695'
                params.subcarrier_cycles_per_etu = 1;
                params.subcarrier_divider = 8;
            case '3390'
                params.subcarrier_cycles_per_etu = 1;
                params.subcarrier_divider = 4;
            case '6780'
                params.subcarrier_cycles_per_etu = 1;
                params.subcarrier_divider = 2;
            otherwise
               error('Unspecified bit rate in PICC_signal_encoding');
         end
    otherwise
        error('Wrong signal interface type specified in PICC_signal_encoding');


end
```

N.12.14 get_subcarrier_states_Type_A_106.m

```
function [bit_states, subcarrier_states, subcarrier_indices, ...
  ignore_last_subcarrier, ret_val] = ...
  get_subcarrier_states_Type_A_106(bb_sig_filt, ...
  samples_per_subcarrier_cycle, subcarrier_indices)

% function [bit_states, subcarrier_states, subcarrier_indices, ...
%  ignore_last_subcarrier, ret_val] = ...
%  get_subcarrier_states_Type_A_106(bb_sig_filt, ...
%  samples_per_subcarrier_cycle, subcarrier_indices)
%
% DESCRIPTION: Function for determining the subcarrier states for a
%              Type A 106 kbps PICC response
%
%              According to ISO/IEC 14443-2:2019 a Type A signal starts
%              with state MS1. For a Type A 106 kbps signal
%              either the first or the second half of each etu is
%              modulated. The etu can be [1,1,1,1,-1,-1,-1,-1] (logic 1)
%              or [-1,-1,-1,-1,1,1,1,1] (logic 0),
%              where a subcarrier state of -1 denotes no subcarrier
%              modulation and a subcarrier state of 1 denotes that the
%              corresponding subcarrier starts with MS1 in the first half
%              and MS2 in the second half.
%
%              For decoding the baseband signal is correlated with one
%              ideal subcarrier period.
%              The extrema of the correlation signal correspond to the
%              modulated part of the etu.
%
% INPUT:
%       bb_sig_filt.....Complex baseband signal
%       samples_per_subcarrier_cycle.....Samples in one subcarrier
%       subcarrier_indices.....Vector containing the subcarrier
%                              boundaries
%
% OUTPUT:
%       bit_states.....States (logical 0b or 1b) of the bits
%                      in the PICC response
%       subcarrier_states......The subcarrier states (0, 1 or -1) in the
%                              PICC response
%       subcarrier_indices.....Vector containing the possibly modified
%                              subcarrier boundaries. If not modified this
%                              vector contains the same elements as the
%                              passed subcarrier boundaries
%       ignore_last_subcarrier.....Flag, 1b indicates that the last
%                                  subcarrier should be ignored
%       ret_val.....Error flag, If not zero, an error occurred


ret_val = 0; % Error indicator for calling function
```

```
samples_per_half_subcarrier = round(samples_per_subcarrier_cycle / 2);

% Define correlator wavelet: One bipolar subcarrier
subcarrier = [ones(samples_per_half_subcarrier, 1); ...
    -ones(samples_per_half_subcarrier, 1)];
wavelet = repmat(subcarrier, 1, 1);

% Add backoff to baseband signal to address correlator fade-in and
% fade-out effects.
backoff = 2 * length(wavelet);

modified_baseband_signal = [bb_sig_filt(1) * ones(backoff, 1);
    bb_sig_filt;
    bb_sig_filt(end) * ones(backoff, 1)];

modified_subcarrier_indices = subcarrier_indices + backoff;
signal_length = length(modified_baseband_signal);



% Evaluate signal power over whole response to determine which signal
% channel (or both channels) should be used for subcarrier state
% determination.
channel_selector = evaluate_signal_power(modified_baseband_signal, ...
    modified_subcarrier_indices);

%----------------Calculation of bit boundaries
bitgrid_indices = modified_subcarrier_indices(1 : 8 : end);

% The response detection algorithm detects the first and last edge of
% subcarrier modulation in the baseband signal. If the last bit has a
% state of logic 1, the 4 subcarriers in the second half are not
% detected.
% In this case the detected response hat to be appended by those 4
% subcarriers.
if  modified_subcarrier_indices(end) - bitgrid_indices(end) >=  ...
        4 * samples_per_subcarrier_cycle

    bitgrid_indices = [bitgrid_indices; bitgrid_indices(end) + ...
        8 * samples_per_subcarrier_cycle];

    modified_subcarrier_indices = (bitgrid_indices(1) : ...
        samples_per_subcarrier_cycle : ...
        bitgrid_indices(end))';

    % Also append baseband signal aligned subcarrier indices
    subcarrier_indices = modified_subcarrier_indices - backoff;

    % If the subcarriers were appended, the last bit possibly has a state
    % of logic 1.
    % In this case the last modulated subcarrier is the (end-5)th
    % subcarrier index.
    % Information about the last modulated subcarrier is needed for
    % filtering the
    % local extrema used for subcarrier detection.
    last_modulated_subcarrier = modified_subcarrier_indices(end - 5);

else
    % In this case the last bit has a possible state of logic 0.
    last_modulated_subcarrier = modified_subcarrier_indices(end - 1);
end



%Offset: First extrema is located half of a subcarrier before the first
%subcarrier. Additional offset of 1/4 subcarrier to address position
%inaccuracies of the extrema
%------------------Determination of bit- and subcarrier states-----------

xc = xcov(modified_baseband_signal, wavelet);
```

```
xc = xc(signal_length : end - backoff - 1);


% -------------- Use both channels of baseband signal
if strcmp(channel_selector, 'BOTH')

    %--------------------Real Part
    [max_idx_re, max_vals_re, min_idx_re, min_vals_re] = ...
        get_extrema_for_state_determination(real(xc), ...
        modified_subcarrier_indices, last_modulated_subcarrier);

    extrema_idx = sortrows([max_idx_re; min_idx_re]);

    [bit_states_re, ~, ignore_last_subcarrier_re] = ...
        bit_state_and_subcarrier_state_detection(extrema_idx, ...
        bitgrid_indices, samples_per_subcarrier_cycle);

    %--------------------Imag Part
    [max_idx_im, max_vals_im, min_idx_im, min_vals_im] = ...
        get_extrema_for_state_determination(imag(xc), ...
        modified_subcarrier_indices, last_modulated_subcarrier);

    extrema_idx = sortrows([max_idx_im; min_idx_im]);

    [bit_states_im, ~, ignore_last_subcarrier_im] = ...
        bit_state_and_subcarrier_state_detection(extrema_idx, ...
        bitgrid_indices, samples_per_subcarrier_cycle);

    ignore_last_subcarrier = ignore_last_subcarrier_re | ...
      ignore_last_subcarrier_im;

    %------------------Merge information from both signals

    [bit_states, unequal_bit_states] = ...
    merge_and_compare_bit_states_from_both_baseband_channels(...
    bit_states_re, bit_states_im);

    if ~isempty(unequal_bit_states)
        str = sprintf(['Deteceted unequeal bit states in folloring ',...
        'bits:\n\t%s'], sptintf('%d ', different_bit_states));
        warning(str)
        ret_val =  -1;
    end


% -------------- Use only imaginary part
elseif strcmp(channel_selector, 'IMAG')

    [max_idx_im, max_vals_im, min_idx_im, min_vals_im] = ...
        get_extrema_for_state_determination(imag(xc), ...
        modified_subcarrier_indices, last_modulated_subcarrier);

    extrema_idx = sortrows([max_idx_im; min_idx_im]);

    [bit_states, ~, ignore_last_subcarrier] = ...
        bit_state_and_subcarrier_state_detection(extrema_idx, ...
        bitgrid_indices, samples_per_subcarrier_cycle);

% -------------- Use only real part
elseif strcmp(channel_selector, 'REAL')

    [max_idx_re, max_vals_re, min_idx_re, min_vals_re] = ...
        get_extrema_for_state_determination(real(xc), ...
        modified_subcarrier_indices, last_modulated_subcarrier);

    extrema_idx = sortrows([max_idx_re; min_idx_re]);

    [bit_states, ~, ignore_last_subcarrier] = ...
        bit_state_and_subcarrier_state_detection(extrema_idx, ...
        bitgrid_indices, samples_per_subcarrier_cycle);
```

```
end

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

if any(bit_states == -1)
    str = sprintf('Invalid bit states found in bits %s.', ...
        sprintf('%d ', find(bit_states == -1)));
    warning(str)
    ret_val = -1;
end

%-------------------------Subcarrier state determination-----------------
% The subcarrier states within each bit are as following:
%
% bit_state == 1 --> subcarrier_states = [1,1,1,1,-1,-1,-1,-1]
% bit_state == 0 --> subcarrier_states = [-1,-1,-1,-1,1,1,1,1]
% bit_state == -1 --> subcarrier_states = [-1,-1,-1,-1,-1,-1,-1,-1]
%

subcarrier_states = -ones(length(bit_states) * 8, 1);

for k = 1 : length(bit_states)
    current_bit_state = bit_states(k);
    subcarrier_idx_start = (k-1) * 8 + 1;

    % Case current_bit_state not handled separately because subcarrier
    % states are -1 per default.
    if current_bit_state == 1
        subcarrier_states(subcarrier_idx_start : ...
        subcarrier_idx_start + 3) = 1;
    elseif current_bit_state == 0
        subcarrier_states(subcarrier_idx_start + 4 : ...
            subcarrier_idx_start + 7) = 1;
    end
end
end




function [bit_states, subcarrier_states, ignore_last_subcarrier] = ...
    bit_state_and_subcarrier_state_detection(extrema_idx, ...
    bitgrid_indices, samples_per_subcarrier_cycle)

% function [bit_states, subcarrier_states, ignore_last_subcarrier] = ...
%    get_subcarrier_states_Type_A_106(extrema_idx, bitgrid_indices, ...
%    samples_per_subcarrier_cycle)
%
% DESCRIPTION: Function for determining the bit- and subcarrier states of
%              the PICC response from the extrema located in each bit
%
%              In an ideal case 4 maxima and 4 minima are located either
%              in the first or the second half of each etu. This is used
%              for determining the bit state. Due to non-ideal signals
%              some extrema can occur outside of the first of second half
%              of half of an etu.
%
%
% INPUT:
%       extrema_idx.....Indices of all extrema in the PICC response
%       bitgrid_indices.....Indices of the bit boundaries
%       samples_per_subcarrier_cycle.....Samples in one subcarrier cycle
%
% OUTPUT:
%       bit_states.....bit states (logic 0b or 1b)
%       subcarrier_states.....The subcarrier states (0, 1 or -1)
%           contained in the PICC response
%       ignore_last_subcarrier.....Flag, 1b indicates that the last
%           subcarrier should be ignored
```

```
% The local extremum that corresponds to the first subcarrier in the
% response is located 1/2 subcarrier before the start this subcarrier.
alignment_offset = round(3 * samples_per_subcarrier_cycle / 4);
correlator_bitgrid_indices = bitgrid_indices - alignment_offset;

N_bits = length(bitgrid_indices) - 1;
bit_boundary_matrix = [correlator_bitgrid_indices(1 : N_bits), ...
    correlator_bitgrid_indices(2 : N_bits + 1)];

bit_states = -ones(N_bits, 1);
subcarrier_states = -ones(8 * N_bits, 1);
ignore_last_subcarrier = 0;

for k = 1 : N_bits
    extrema_in_bit = extrema_idx((extrema_idx > ...
        bit_boundary_matrix(k, 1)) ...
        & (extrema_idx < bit_boundary_matrix(k, 2)));

    %Determine extrema in first and second half
    extrema_in_first_half = extrema_in_bit(...
        (extrema_in_bit > bit_boundary_matrix(k, 1)) & ...
        (extrema_in_bit < bit_boundary_matrix(k, 1) + ...
        4 * samples_per_subcarrier_cycle));

    extrema_in_second_half = extrema_in_bit(...
        (extrema_in_bit > bit_boundary_matrix(k, 1) + ...
        4 * samples_per_subcarrier_cycle) & ...
        (extrema_in_bit < bit_boundary_matrix(k, 2)));


    % In case of an ideal signal 8 extrema are located in either the
    % first or the second half of each bit.
    %
    % In some cases one or two additional extrema can occur within 1.5
    % subcarriers after (or before; bit state dependent) the half of the
    % bit due to the signal shape.
    % In such case no error is detected.


    % 8 extrema in first half (logic 1)
    if ismember(length(extrema_in_first_half), [7,8])
        % No extrema in second half
        if isempty(extrema_in_second_half)
            bit_states(k) = 1;
        % One or two extrema in second half
        elseif length(extrema_in_second_half) <= 2
            % If those extrema are within 1.5 subcarriers after the
            % boundary of the first half the bit state can still be
            % detected
            if all((extrema_in_second_half < bit_boundary_matrix(k, 1) ...
                + 5.5 * samples_per_subcarrier_cycle) | ...
               (extrema_in_second_half > bit_boundary_matrix(k, 1) + ...
               6.5 * samples_per_subcarrier_cycle))

                bit_states(k) = 1;
            % If not, bit state of the bit could not be detected
            else
                bit_states(k) = -1;
            end
        end


    % Same as above but for logic 0
    elseif ismember(length(extrema_in_second_half), [7,8])
        % No extrema in first half
        if isempty(extrema_in_first_half)
            bit_states(k) = 0;
        % One or two extrema in first half
        elseif length(extrema_in_first_half) <= 2
            % If those extrema are within 1.5 subcarriers before the
            % lower boundary of the second half the bit state can still
```

```
            % be detected
            if all((extrema_in_first_half > ...
              bit_boundary_matrix(k, 1) + ...
               3.5 * samples_per_subcarrier_cycle) | ...
              (extrema_in_first_half < bit_boundary_matrix(k, 1) + ...
               1.5 * samples_per_subcarrier_cycle))

                bit_states(k) = 0;
            % If not, bit state of the bit could not be detected
            else
                bit_states(k) = -1;
            end
        end

        % Next two cases handle to exclude the last subcarrier
    elseif ismember(length(extrema_in_first_half), [5,6]) && k == N_bits
        % No extrema in second half
        if isempty(extrema_in_second_half)
            bit_states(k) = 1;
            ignore_last_subcarrier = 1;
            %One or two extrema in second half
        elseif length(extrema_in_second_half) <= 2
            % If those extrema are within 1.5 subcarriers after the
            % boundary of the first half the bit state can still be
            % detected
            if all((extrema_in_second_half < ...
              bit_boundary_matrix(k, 1) + ...
              5.5 * samples_per_subcarrier_cycle) | ...
              (extrema_in_second_half > bit_boundary_matrix(k, 1) + ...
              6.5 * samples_per_subcarrier_cycle))

                bit_states(k) = 1;
                ignore_last_subcarrier = 1;
                % If not, bit state of the bit could not be detected
            else
                bit_states(k) = -1;
            end

        end
    % Same as above but for logic 0
    elseif ismember(length(extrema_in_second_half), [5,6]) && k == N_bits
        % No extrema in first half
        if isempty(extrema_in_first_half)
            bit_states(k) = 0;
            ignore_last_subcarrier = 1;
            % One or two extrema in first half
        elseif length(extrema_in_first_half) <= 2
            % If those extrema are within 1.5 subcarriers before the
            % lower boundary of the second half the bit state can still
            % be detected
            if all((extrema_in_first_half > ...
              bit_boundary_matrix(k, 1) + ...
              3.5 * samples_per_subcarrier_cycle) | ...
              (extrema_in_first_half < bit_boundary_matrix(k, 1) + ...
              1.5 * samples_per_subcarrier_cycle))

                bit_states(k) = 0;
                ignore_last_subcarrier = 1;

                % If not, bit state of the bit could not be detected
            else
                bit_states(k) = -1;
            end
        end
    end
end

% When the last subcarrier in the response shall be ignored, remove it
% from the subcarrier states vector
if ignore_last_subcarrier
    subcarrier_states = subcarrier_states(1 : end - 1);
```

```
end

end



function [max_idx, max_vals, min_idx, min_vals] = ...
    get_extrema_for_state_determination(correlation_signal, ...
    subcarrier_indices, last_modulated_subcarrier)

% function [max_idx, max_vals, min_idx, min_vals] = ...
%    get_extrema_for_state_determination(correlation_signal, ...
%    subcarrier_indices, samples_per_subcarrier_cycle)
%
% DESCRIPTION: Function for determining the extrema used for bit- and
%              subcarrier state determination from the passed correlation
%              signal
%
%
% INPUT:
%       correlation_signal.....Result of cross correlation of the baseband
%                                       signal with one subcarrier cycle
%       subcarrier_indices.....Vector containing the subcarrier boundaries
%       last_modulated_subcarrier.....Index of the begin of the last
%           modulated subcarrier in the response
%
% OUTPUT:
%       max_idx......Indices of the local maxima
%       max_vals.....Values of the local maxima
%       min_idx......Indices of the local minima
%       min_vals.....Values of the local minima

%Enable (1) or disable (0) the plot functionality in this function
PLOT = 0;
samples_per_subcarrier_cycle = subcarrier_indices(2) - ...
  subcarrier_indices(1);

% The local extremum that corresponds to the first subcarrier in the
% response is located 1/2 subcarrier before the start this subcarrier.
alignment_offset = round(3 * samples_per_subcarrier_cycle / 4);
correlator_subcarrier_indices = subcarrier_indices - alignment_offset;
last_modulated_subcarrier = last_modulated_subcarrier - alignment_offset;

sig = correlation_signal;
[max_vals, max_idx] = new_findpeaks(sig);

sig = correlation_signal;
[min_vals, min_idx] = new_findpeaks(-sig);


% Remove extrema before the first subcarrier and after the last
% subcarrier.
% As the first extrema occurs one half subcarrier before the first
% subcarrier, the offset and a tolerance of 1/4 subcarrier are added.

offset = round(3 * samples_per_subcarrier_cycle / 4);
max_vals(max_idx < correlator_subcarrier_indices(1) - offset) = [];
max_idx(max_idx < correlator_subcarrier_indices(1) - offset) = [];

min_vals(min_idx < correlator_subcarrier_indices(1) - offset) = [];
min_idx(min_idx < correlator_subcarrier_indices(1) - offset) = [];

max_vals(max_idx > correlator_subcarrier_indices(end)) = [];
max_idx(max_idx > correlator_subcarrier_indices(end)) = [];

min_vals(min_idx > correlator_subcarrier_indices(end)) = [];
min_idx(min_idx > correlator_subcarrier_indices(end)) = [];
min_vals = -min_vals;
```

```
% Calculate peak envelope of the correlation to address amplitude changes
% caused by phase drift
[env_up, env_lo] = peak_envelope(correlation_signal, 1, ...
  length(correlation_signal));

[env_up, env_lo, ~] = envelope_postprocessing(env_up, env_lo, ...
    subcarrier_indices, 2 * samples_per_subcarrier_cycle, max_idx, ...
    max_vals, min_idx, min_vals);

threshold = abs(env_up - env_lo);
threshold = threshold / max(threshold);

% Bitrate dependent post processing
threshold(1 : correlator_subcarrier_indices(1)) = 1.1;
idx = last_modulated_subcarrier + samples_per_subcarrier_cycle + ...
    (1 : samples_per_subcarrier_cycle);
threshold(idx) = 1;
threshold(correlator_subcarrier_indices(end) : end) = 1.1;


% Scaling factor was empirically determined
scaling_factor = 0.6;

max_xc = max(correlation_signal);
min_xc = min(correlation_signal);

% Find the detected extrema by using the modified peak envelopes as a
% dynamic threshold
max_idx_ = max_idx;
max_idx(max_vals < scaling_factor * max_xc * threshold(max_idx_)) = [];
max_vals(max_vals < scaling_factor * max_xc * threshold(max_idx_)) = [];


min_idx_ = min_idx;
min_idx(min_vals > scaling_factor * min_xc * threshold(min_idx_)) = [];
min_vals(min_vals > scaling_factor * min_xc * threshold(min_idx_)) = [];

if PLOT

figure()
plot(correlation_signal)
hold on
plot(max_idx, max_vals, 'ro')
plot(min_idx, min_vals, 'go')
plot(threshold * scaling_factor * max_xc)
plot(threshold * scaling_factor * min_xc)
plot(correlator_subcarrier_indices(:) * [1,1], ...
    [max(correlation_signal), min(correlation_signal)], 'Color', 'black')
grid minor
xlabel('Samples')
title('Correlation signal with local extremas used for state detection')
legend('Correlation signal', 'Local maxima', 'Local minima', ...
    'Upper threshold', 'Lower threshold', 'Subcarrier boundaries')

end

end


function [bit_states, unequal_bit_states] = ...
    merge_and_compare_bit_states_from_both_baseband_channels( ...
    bit_states_re, bit_states_im)

% function [bit_states, unequal_bit_states] = ...
%    merge_and_compare_bit_states_from_both_channels(
%    bit_states_re, bit_states_im)
%
% DESCRIPTION: If both baseband channels are used for bit- and subcarrier
%   state determination, this function is used to merge the bit-state
%   results from both channels
%
```

```
%
%   The information is merged as following:
%       -) If both channels contain valid bit states (0 or 1) and the bit
%          states in both channels are equal bit_states(k) is set to the
%          corresponding value.
%
%          If the valid bit states are unequal, the index of the
%          corresponding bit is added to unequal_bit_states. If this
%          vector is returned empty the merge of the information was
%          successful. Otherwise an error occurred.
%
%       -) If one channel delivers bit state -1 and the other delivers a
%          valid bit state (0 or 1), bit_states(k) is set to the valid
%          bit state
%       -) If both channels deliver a bit state of -1 bit_states(k) is
%          set to -1
%
%
% INPUT:
%       bit_states_re..... Bit states from real channel
%       bit_states_im..... Bit states from imaginary channel
%
% OUTPUT:
%       bit_states..... Vector containing the final bit states.
%       unequal_bit_states..... Vector containing the indices of bits
%       with different valid bit-states in both channels.

unequal_bit_states = [];

    bit_states = -ones(length(bit_states_re), 1);
    for k = 1 : length(bit_states_re)
        bit_state_re = bit_states_re(k);
        bit_state_im = bit_states_im(k);

        %Valid bit state in both signals detected
        if bit_state_re ~= -1 && bit_state_im ~= -1
            % For valid resulting bit state, bit state of both signals
            % must be equal
            if bit_state_re == bit_state_im
                bit_states(k) = bit_state_re;
                % Error: Bit states from real and imaginary part are
                % different
            else
                bit_states(k) = -1;
                unequal_bit_states = [unequal_bit_states; k];
            end

            % At least one signal contains invalid bit state
            if bit_state_re == -1 && bit_state_im ~= -1
                bit_states(k) = bit_state_im;
            elseif bit_state_re ~= -1 && bit_state_im == -1
                bit_states(k) = bit_state_re;
            end
        end
    end

end
```

N.12.15 get_subcarrier_states_Type_B_A_not_106.m

```
function [subcarrier_states, subcarriers_modified, ...
    subcarrier_indices, ignore_last_subcarrier, ...
    state_determination_error] = ...
    get_subcarrier_states_Type_B_A_not_106(baseband_signal, ...
    subcarrier_indices, encoding)

% function [subcarrier_states, subcarriers_modified, ...
%    subcarrier_indices, ignore_last_subcarrier, ...
%    state_determination_error] = ...
%    get_subcarrier_states_Type_B_A_not_106(baseband_signal, ...
```

```
%    subcarrier_indices, encoding)
%
% DESCRIPTION: Function for determining the subcarrier states for Type B
%              all bit rate signals and Type A higher than 106kbps signals
%
%            A subcarrier state equal to "1" corresponds to a subcarrier
%            with state MS1 in the first half and MS2 in the second half.
%            A subcarrier state equal to "0" corresponds to a subcarrier
%            with state MS2 in the first half and MS1 in the second half.
%
%            The function computes the cross-correlation between the real
%            and imaginary part of the complex baseband signal and a
%            sequence of one subcarrier and one inverse subcarrier which
%            is equal to the shape of the baseband signal in case of a
%            phase transition as well as with one ideal subcarrier period.
%
%            In case of Type B signals the reference subcarrier boundaries
%            might be shifted by half of a subcarrier because
%            non-integer number of subcarrier cycles within TR1. In this
%            case it is possible that the start detection algorithm
%            detects the transition between MS2 and MS1 in the first
%            subcarrier cycle and not the real start of the response.
%
%
%            For Type A signals the PICC response starts with a subcarrier
%            sequence of state 1.
%
% INPUT:
%        baseband_signal.....Filtered complex baseband signal
%        subcarrier_indices.....Subcarrier boundaries
%        encoding.....Structure containing the encoding of the signal
%
% OUTPUT:
%       subcarrier_states.....The determines states of each subcarrier in
%                             the PICC response
%       subcarriers_modified.....Flag (0 or 1) indicating if the
%                                subcarrier boundaries passed to the
%                                function were shifted (only relevant for
%                                Type B signals)
%       subcarrier_indices.....The modified subcarrier boundaries
%       ignore_last_subcarrier.....Flag denoting if the last detected
%                                  might not belong to the PICC response
%       state_determination_error......If unequal to zero, an error
%                                      occurred.
%

subcarrier_states = [];
subcarriers_modified = 0;
state_determination_error = 0;
ignore_last_subcarrier = 0;

%------------------------Signal power evaluation-------------------------
% Calculate signal powers of real and imaginary part of PICC response and
% decide if one of the parts can be ignore due to low signal energy

%0 : Take both signals, 1: Ignore real part, 2 : Ignore imaginary part
channel_selector = evaluate_signal_power(baseband_signal, ...
    subcarrier_indices);

switch channel_selector
    % When both channels are used, the subcarrier states are determines
    % using the phase transitions. This is done because the subcarrier
    % state determination using single subcarrier correlation in case of
    % interstate phase-drifts larger than 90 ° results in an information
    % change from the real to the imaginary part and vice versa.
    %
    % The state relation between the real- and imaginary channel can be
    % either in-phase or inverse-phase. This means that the states
    % determined for the real channel can be 00001110100 and for the
    % imaginary channel 11110001011. When the PICC response contains an
    % interstate phase-drift of more than 90 degrees
```

```
% the state relation can change over the response, typically every
% 90 degrees.
case 'BOTH'
    fprintf(['Due to the signal power ratio between the real ', ...
        'and imaginary channels\nof the complex baseband signal, ', ...
        'both baseband channels are analyzed.\n\r'])
    fprintf(['The output below belongs to the real channel of ', ...
        'the PICC response.\n\r'])
    % ----- Real channel
    [~, subcarrier_indices_re, initial_subcarrier_state_re, ...
        return_value_re, ~, ...
        phase_transition_data_re] = ...
    single_channel_subcarrier_state_determination(...
        real(baseband_signal), ...
        subcarrier_indices, encoding);

    fprintf(['The output below belongs to the imaginary channel ', ...
        'of the PICC response.\n\r'])
    % ----- Imaginary channel
    [~, subcarrier_indices_im, initial_subcarrier_state_im, ...
        return_value_im, ~, ...
        phase_transition_data_im] = ...
    single_channel_subcarrier_state_determination(...
        imag(baseband_signal), ...
        subcarrier_indices, encoding);

    % ----- Error handling
    % Print error messages depending on the returned error codes from
    % each channel
    state_determination_error = dual_channel_error_handling( ...
        return_value_re, return_value_im, ...
        initial_subcarrier_state_re, initial_subcarrier_state_im, ...
        subcarrier_indices_re, subcarrier_indices_im);

    if state_determination_error
        return
    end

    % ----- Subcarrier state determination
    initial_subcarrier_state = initial_subcarrier_state_re;

    number_of_subcarriers = length(subcarrier_indices_re) - 1;

    subcarrier_states = ...
        determine_subcarrier_states_from_phase_transitions( ...
        phase_transition_data_re, phase_transition_data_im, ...
        initial_subcarrier_state, number_of_subcarriers);

    % ---- Use the mean value between the two subcarrier grids
    subcarrier_indices = round( ...
        mean([subcarrier_indices_re, subcarrier_indices_im], 2));

case 'REAL'
    [subcarrier_states, subcarrier_indices, ~, ...
        return_value, ~, ~] = ...
     single_channel_subcarrier_state_determination(...
        real(baseband_signal), ...
        subcarrier_indices, encoding);

    if return_value ~= 0
        state_determination_error = true;
        return
    end

case'IMAG'
    [subcarrier_states, subcarrier_indices, ~, ...
        return_value, ~, ~] = ...
        single_channel_subcarrier_state_determination(...
        imag(baseband_signal), ...
        subcarrier_indices, encoding);
```

```
        if return_value ~= 0
            state_determination_error = true;
            return
        end
end



end



function [subcarrier_states, subcarrier_indices, ...
    initial_subcarrier_state, return_value, ...
    low_amplitude_subcarriers, phase_transition_data] = ...
    single_channel_subcarrier_state_determination(baseband_channel, ...
    subcarrier_indices, encoding)

% function [subcarrier_states, subcarrier_indices, ...
%    initial_subcarrier_state, return_value, ...
%    low_amplitude_subcarriers, phase_transition_data] = ...
%    single_channel_subcarrier_state_determination(baseband_channel, ...
%    subcarrier_indices, encoding)
%
%
% DESCRIPTION: Function for determining the states of the subcarriers
%     contained in the response.
%
%     ATTENTION: READ THIS SECTION CAREFULLY:
%     ---------------------------------------------------------------
%     The state of the first subcarrier (initial_subcarrier_state)
%     determines whether the first or the second half of each subcarrier
%     belongs to MS1 or MS2.
%
%     The function determine_subcarrier_states_from_correlation()
%     determines, the subcarrier states from the position and type of the
%     local extrema of the single subcarrier correlation signal within
%     each subcarrier.
%
%     The state computed in
%     "determine_subcarrier_states_from_correlation()"
%     might be inverse to the baseband subcarrier states.
%     Therefore, this function evaluates as a last step if the first
%     determined subcarrier state from
%     determine_subcarrier_states_from_correlation()
%     is equal to the initial subcarrier state. If not, it changes ('0'
%     to '1' and '1' to '0') the states.
%
%     If the first subcarrier is a subcarrier with a low amplitude
%     (low_amplitude_subcarriers(1) == 1) the function is not able to
%     determine the final subcarrier states.
%     In this case the function returns the states computed by
%     determine_subcarrier_states_from_correlation() without modification
%     and sets the return_value to 1.
%     Otherwise return_value is set to -1 because the final subcarrier
%     states could not be determined.
%
%     If, due to the signal power, only one channel of the complex
%     baseband signal is analyzed, the subcarrier state determination
%     failed.
%
%     If both channels are used, the other channel can be used to
%     determine if the states returned by this function has to be
%     inverted or not.
%     ---------------------------------------------------------------
%
%
% INPUT:
%   baseband_channel.....Baseband channel whose subcarrier state should
%       be determined
%   subcarrier_indices.....Vector of subcarrier boundaries
```

```
%   encoding.....Structure specifying the encoding of the response
%
%
% OUTPUT:
%   subcarrier_states.....Vector containing a first estimate of the
%       subcarrier states
%   subcarrier_indices.....Vector fo modified subcarrier indices
%   initial_subcarrier_state.....When
%   return_value.....Value giving information about the success of the
%       state determination. Can have one of the following values:
%           -)  0: Success
%           -) -1: Error
%           -)  1: Could not determine final subcarrier states, because
%                  first subcarrier in response has a low amplitude
%    low_amplitude_subcarriers...
%    phase_transition_data...

return_value = 0;
subcarrier_states = [];
phase_transition_data = [];

PLOT = 0;

N_bb = length(baseband_channel);
samples_per_subcarrier = subcarrier_indices(2) - subcarrier_indices(1);

% This function assigns an initial subcarrier state (MS1/MS2 state) which
%  will be updated according to the input signals properties.
%
% Type A PICC responses handled in this function always have an initial
% subcarrier state of 1.
% Type B PICC responses have an initial subcarrier state of 1 if NO
% subcarrier shift (see later) has to be applied. Otherwise the response
% has an intial subcarrier state of 0.
initial_subcarrier_state = 1;

if strcmp(encoding.interface_type, 'Type_B')
    Type_B = 1;
else
    Type_B = 0;
end

% ------------------ Single subcarrier correlation ----------------------
%
% Correlate the baseband channel with one bipolar subcarrier. Afterwards,
% calculate the upper and lower peak envelopes from the resulting
% correlation signal.
%
% Those envelopes are then smoothed with a moving average filter further
% modified for usage in further calculation

% ----- Perform single subcarrier correlation
xc = single_subcarrier_correlation(baseband_channel, ...
    samples_per_subcarrier);

% ----- Calculate local extrema of subcarrier correlation signal
[corr_max_idx, corr_max_vals, corr_min_idx, corr_min_vals] = ...
    determine_single_subcarrier_correlation_extrema(xc, ...
    subcarrier_indices);

% ----- Compute upper and lower peak envelopes
% Peak envelope is calculated for whole correlation signal. The
% smoothing-filter length (moving average) is 2 * samples_per_subcarrier
[env_up, env_lo] = peak_envelope(xc, 1, N_bb);


% ----- Envelope post-processing
filter_length = 2 * samples_per_subcarrier;
[env_up_filt, env_lo_filt, low_amplitude_subcarriers] = ...
    envelope_postprocessing( env_up, env_lo, subcarrier_indices, ...
    filter_length, corr_max_idx, corr_max_vals, corr_min_idx, ...
```

```
    corr_min_vals);



% ----- Remove extrema with low amplitudes
[corr_max_idx, corr_max_vals, corr_min_idx, corr_min_vals] = ...
    single_subcarrier_correlation_extrema_postprocessing( ...
    corr_max_idx, corr_max_vals, corr_min_idx, corr_min_vals, ...
    env_up_filt, env_lo_filt);


if PLOT
    plot_peak_envelopes(xc, env_up, env_lo, env_up_filt, env_lo_filt, ...
        subcarrier_indices)

     plot_single_subcarrier_correlation_results(baseband_channel, xc, ...
        env_up_filt, env_lo_filt, subcarrier_indices, corr_max_idx, ...
        corr_max_vals, corr_min_idx, corr_min_vals);
end


% ------------------- Phase transition correlation ----------------------
% Correlate the baseband channel with a subcarrier phase transition
% sequence (regular bipolar subcarrier + inverse bipolar subcarrier).
% Using the peak envelopes calculated above as dynamic threshold, the
% phase transitions are determined via the points of maximum correlation
% in the baseband channel.
%
% Additionally it is determined if the subcarrier grid has to be shifted
%  back by 1/2 subcarrier.

% ----- Correlate baseband channel with phase transition sequence
xc = phase_transition_correlation(baseband_channel, ...
    samples_per_subcarrier);

% ----- Determine and filter local extrema from resulting signal
% Each determined local extrema corresponds to a subcarrier phase
% transition

[phase_trans_max_idx, ~, phase_trans_min_idx, ~] = ...
    determine_phase_transition_correlation_extrema(xc, ...
    subcarrier_indices, env_up_filt, env_lo_filt, encoding);

% ----- Check if distances between phase transitions are integer numbers
% of subcarriers
check_passed = phase_transition_distance_check(...
    phase_trans_max_idx, phase_trans_min_idx, samples_per_subcarrier);

if ~check_passed
    warning(['Phase transitions with non integer number of ', ...
        'subcarrier distances ', 'detected. \nTerminating script.'])

    return_value = -1;
    return
end

% ----- Check if no consecutive phase transitions of the same type
% occurred
check_passed = phase_transition_sequence_check(phase_trans_max_idx, ...
    phase_trans_min_idx, low_amplitude_subcarriers, subcarrier_indices);

if ~check_passed
    warning(['Consecutive phase transitions of same type detected.', ...
        '\nTeminating script'])

    return_value = -1;
    return
end

% ----- Check if a subcarrier shift for a Type B response is required
```

```
if Type_B
    [shift_subcarriers, error] = Type_B_subcarrier_shift(...
        phase_trans_max_idx, phase_trans_min_idx, subcarrier_indices, ...
        encoding);

    if error
        return_value = -1;
        return
    end

    if shift_subcarriers
        subcarrier_indices = subcarrier_indices - ...
        samples_per_subcarrier / 2;

        subcarrier_indices = [subcarrier_indices;
            subcarrier_indices(end) + samples_per_subcarrier];

        low_amplitude_subcarriers = [low_amplitude_subcarriers;
            low_amplitude_subcarriers(end)];

        % In case of a subcarrier shift, the Type B response has an
        % initial subcarrier state of 0
        initial_subcarrier_state = 0;
    end
end


% ----- Assign local extrema to phase transitions
phase_transition_data = assign_phase_transition_extrema_to_subcarriers(...
    phase_trans_max_idx, phase_trans_min_idx, subcarrier_indices);


% ----- Correct subcarrier grid
subcarrier_indices = subcarrier_grid_correction(phase_transition_data, ...
    subcarrier_indices);

% ----- Nominal phase transition check
check_passed = nominal_phase_transition_position_check( ...
    phase_transition_data, subcarrier_indices);

if ~check_passed
    warning(['Not all phase transitions occur at nominal positions. ', ...
        '\nTerminating script.'])

    return_value = -1;
    return
end


% --------------------- Determine subcarrier states ---------------------
[subcarrier_states, remove_last_subcarrier_1, ...
    state_determination_error] = ...
    determine_subcarrier_states_from_single_subcarrier_correlation( ...
    corr_max_idx, corr_min_idx, subcarrier_indices, ...
    low_amplitude_subcarriers);

if state_determination_error
    return_value = -1;
    return
end

% ----- Check if same phase transitions are determines from both
%   correlation signals
[check_passed, remove_last_subcarrier_2] = ...
    phase_transition_comparision_check(phase_transition_data, ...
    subcarrier_states);

if ~check_passed
    warning(['Different phase transitions detected from both ', ...
        'correlation signals.\nTerminating script.'])
    return_value = -1;
```

```
end

% ----- Remove last subcarrier from response if required
if remove_last_subcarrier_1 || remove_last_subcarrier_2
    subcarrier_states = subcarrier_states(1 : end - 1);
    subcarrier_indices = subcarrier_indices(1 : end - 1);
end

% ----- Determine final subcarrier states depending on the initial
% subcarrier state
if subcarrier_states(1) ~= initial_subcarrier_state
    % Could not finally determine subcarrier states due to first
    % subcarrier with low amplitude
    if subcarrier_states(1) == -1 && low_amplitude_subcarriers(1) == 1
        return_value = 1;


    % Set states '0' to '1' and vice versa
    else
        tmp = subcarrier_states;
        subcarrier_states(tmp == 1) = 0;
        subcarrier_states(tmp == 0) = 1;
        clear tmp;
    end
end



end


% ============= Single subcarrier processing functions ==================
function xc = single_subcarrier_correlation(baseband_channel, ...
    samples_per_subcarrier)

% function xc = single_subcarrier_correlation(baseband_signal,
%   samples_per_subcarrier)
%
% DESCRIPTION: Function for performing correlation with one bipolar
%   subcarrier
%
% INPUT:
%   baseband_signal_channel.....One channel of the complex baseband signal
%   samples_per_subcarrier.....Samples in one subcarrier
%
% OUTPUT:
%       xc.....Mean-free cross correlation between one bipolar subcarrier
%          and the passed baseband signal channel
%


% ----- Add backoff to start and end of signal
N_backoff = 2 * samples_per_subcarrier;

modified_baseband_channel = [baseband_channel(1) * ones(N_backoff, 1); ...
    baseband_channel;
    baseband_channel(end) * ones(N_backoff, 1)];


wavelet = [ones(samples_per_subcarrier / 2, 1);
    -ones(samples_per_subcarrier / 2, 1)];

xc = xcov(modified_baseband_channel, wavelet);
N_sig = length(modified_baseband_channel);

%Select upper half of correlation signal and remove backoff
xc = xc(N_sig + N_backoff  : end - N_backoff);
end


function [max_idx, max_vals, min_idx, min_vals] = ...
```

```
    determine_single_subcarrier_correlation_extrema(xc, ...
    subcarrier_indices)

% function [max_idx, max_vals, min_idx, min_vals] = ...
%    determine_single_subcarrier_correlation_extrema(xc, ...
%    subcarrier_indices)
%
%
% DESCRIPTION: Function for determining the local extrema from the
%   resulting signal from correlating the baseband channel with a single
%   bipolar subcarrier
%
%
% INPUT:
%   xc.....Correlation signal from single_subcarrier_correlation()
%   subcarrier_indices.....Vector of subcarrier boundaries
%
%
% OUTPUT:
%       max_idx.....Vector of indices of local maxima in correlation
%           signal
%       max_vals.....Vector of values of local minima in correlation
%           signal
%       min_idx.....Vector of indices of local maxima in correlation
%           signal
%       min_vals.....Vector of values of local minima in correlation
%           signal
%

samples_per_subcarrier = subcarrier_indices(2) - subcarrier_indices(1);

% Shift subcarrier boundaries by 3/4 of a subcarrier to align the
% correlation peak/extrema to the baseband signal
correlator_subcarrier_indices = subcarrier_indices - ...
    round( 3/4 * samples_per_subcarrier );


[max_vals, max_idx, min_vals, min_idx] = ...
    find_and_cleanup_peaks(xc, correlator_subcarrier_indices(1), ...
    correlator_subcarrier_indices(end));
end

function [max_idx, max_vals, min_idx, min_vals] = ...
    single_subcarrier_correlation_extrema_postprocessing(max_idx, ...
    max_vals, min_idx, min_vals, env_up, env_lo)

% function [max_idx, max_vals, min_idx, min_vals] = ...
%   single_subcarrier_correlation_extrema_postprocessing(max_idx, ...
%   max_vals, min_idx, min_vals, env_up, env_lo)
%
%
% DESCRIPTION: Function for removing invalid extrema from the extrema
%               detected in
%               determine_single_subcarrier_correlation_extrema()
%
%
% INPUT:
%   max_idx.....Vector of indices of local maxima in correlation signal
%   max_vals.....Vector of values of local minima in correlation signal
%   min_idx.....Vector of indices of local maxima in correlation signal
%   min_vals.....Vector of values of local minima in correlation signal
%   env_up.....Upper peak envelope of the correlation signal
%   env_lo.....Lower peak envelope of the correlation signal
%
%
% OUTPUT:
%   max_idx.....Vector of indices of local maxima in correlation signal
%   max_vals.....Vector of values of local minima in correlation signal
%   min_idx.....Vector of indices of local maxima in correlation signal
%   min_vals.....Vector of values of local minima in correlation signal
%
```

```
threshold_scaling = 0.5;

max_idx_ = max_idx;
max_idx(max_vals < threshold_scaling * env_up(max_idx_)) = [];
max_vals(max_vals < threshold_scaling * env_up(max_idx_)) = [];


min_idx_ = min_idx;
min_idx(min_vals > threshold_scaling * env_lo(min_idx_)) = [];
min_vals(min_vals > threshold_scaling * env_lo(min_idx_)) = [];


end


function plot_single_subcarrier_correlation_results(baseband_channel, ...
    xc, env_up, env_lo, subcarrier_indices, max_idx, max_vals, ...
    min_idx, min_vals)

% function plot_single_subcarrier_correlation_results(baseband_channel,...
%    xc, env_up, env_lo, subcarrier_indices, max_idx, max_vals, ...
%    min_idx, min_vals)
%
%
% DESCRIPTION: Function for plotting the results of the single subcarrier
%     correlation
%
%
% INPUT:
%   baseband_channel.....Baseband channel used for single subcarrier
%       correlation
%   xc.....Single subcarrier cross-correlation signal
%   env_up.....Upper peak envelope of the correlation signal
%   env_lo.....Lower peak envelope of the correlation signal
%   subcarrier_indices.....Vector of subcarrier boundaries
%   max_idx.....Vector of indices of local maxima in correlation signal
%   max_vals.....Vector of values of local minima in correlation signal
%   min_idx.....Vector of indices of local maxima in correlation signal
%   min_vals.....Vector of values of local minima in correlation signal
%

% Threshold scaling factor from
% determine_single_subcarrier_correlation_extrema()
threshold_scaling = 0.5;

samples_per_subcarrier = subcarrier_indices(2) - subcarrier_indices(1);
correlator_subcarrier_indices = subcarrier_indices - ...
    round( 3/4 * samples_per_subcarrier);

figure()
fprintf(['Plot functionality in ', ...
        'single_channel_subcarrier_state_determination ', ...
        'enabled. \n\tFigure %d shows the correlation signal with ', ...
        'the detected local extrema and the threshold used for ', ...
        'determination of valid extrema.\nAdditionally the ', ...
        'correspoding baseband channel is plotted.\n\r'], ...
        get(gcf, 'Number'))
sig = baseband_channel;
p1 = subplot(211);
plot(sig);
hold on
line(subcarrier_indices * [1,1], [min(sig), max(sig)], 'Color', [0,0,0])
title('Baseband channel')


height = max(sig) + 0.075 * abs(max(sig) - min(sig));
for k = 1 : length(subcarrier_indices) - 1
    start_idx = subcarrier_indices(k);
    end_idx = subcarrier_indices(k + 1);
```

```
    pos_x = mean([start_idx, end_idx]);

    text(pos_x, height, num2str(k), 'Color', [0,0,0], ...
      'HorizontalAlignment', 'center')
end
height_up = max(sig) + 0.15 * abs(max(sig) - min(sig));
height_lo = min(sig) - 0.1 * abs(max(sig) - min(sig));
ylim([height_lo, height_up])


p2 = subplot(212);
plot(xc);
hold on
plot(threshold_scaling * env_up)
plot(threshold_scaling * env_lo)
line(correlator_subcarrier_indices * [1,1], [min(xc), max(xc)], ...
    'Color', [0,0,0])
plot(max_idx, max_vals, 'ro')
plot(min_idx, min_vals, 'go')
title('Single subcarrier correlation signal')


height = max(xc) + 0.075 * abs(max(xc) - min(xc));
for k = 1 : length(correlator_subcarrier_indices) - 1
    start_idx = correlator_subcarrier_indices(k);
    end_idx = correlator_subcarrier_indices(k + 1);

    pos_x = mean([start_idx, end_idx]);

    text(pos_x, height, num2str(k), 'Color', [0,0,0], ...
        'HorizontalAlignment', 'center')
end

linkaxes([p1, p2], 'x')
height_up = max(xc) + 0.15 * abs(max(xc) - min(xc));
height_lo = min(xc) - 0.1 * abs(max(xc) - min(xc));
ylim([height_lo, height_up])

end


function plot_peak_envelopes(xc, env_up, env_lo, env_up_filt, ...
  env_lo_filt, subcarrier_indices)

% function plot_peak_envelopes(xc, env_up, env_lo, env_up_filt, ...
%  env_lo_filt, subcarrier_indices)
%
%
% DESCRIPTION: Function for plotting the peak envelopes used as a dynamic
%    threshold
%
%
% INPUT:
%   baseband_channel.....Baseband channel used for single subcarrier
%     correlation
%   xc.....Single subcarrier cross-correlation signal
%   env_up.....Upper peak envelope of the correlation signal
%   env_lo.....Lower peak envelope of the correlation signal
%   env_filt.....Postprocessed upper peak envelope of the correlation
%     signal
%   env_filt.....Postprocessed lower peak envelope of the correlation
%     signal
%   subcarrier_indices.....Vector of subcarrier boundaries
%

    samples_per_subcarrier = subcarrier_indices(2) - ...
      subcarrier_indices(1);

    figure()
    fprintf(['Plot functionality in ', ...
      'single_channel_subcarrier_state_determination()','enabled. ', ...
```

```
      '\n\tFigure %d shows the passed signal with its upper ', ...
      'and lower envelopes.\n\r'], get(gcf, 'Number'))
    plot(xc)
    hold on
    %plot(env_up_filt)
    %plot(env_lo_filt)
    plot(env_up)
    plot(env_lo)
    title('Peak envelopes')
    grid minor
    xlabel('Samples')
    height = max(xc) + 0.1 * abs(max(xc) - min(xc));
    line((subcarrier_indices - round(samples_per_subcarrier * 3/4)) * ...
        [1,1], [height, min(xc)], 'Color', [0,0,0])
    legend('Signal', 'Filtered Upper envelope', ...
        'Filtered Lower envelope', 'Unfitered upper envelope', ...
        'Unfiltered lower envelope')
    height_up = max(xc) + 0.15 * abs(max(xc) - min(xc));
    height_lo = min(xc) - 0.1 * abs(max(xc) - min(xc));
    ylim([height_lo, height_up])

end

function [subcarrier_states, remove_last_subcarrier, ...
    state_determination_error] = ...
  determine_subcarrier_states_from_single_subcarrier_correlation( ...
  max_idx, min_idx, subcarrier_indices, low_amplitude_subcarriers)

state_determination_error = false;

samples_per_subcarrier = subcarrier_indices(2) - subcarrier_indices(1);
correlator_subcarrier_indices = subcarrier_indices - ...
    round( 3/4 * samples_per_subcarrier);

N_subcarriers = length(subcarrier_indices) - 1;
subcarrier_states = -ones(N_subcarriers, 1);

extrema_in_subcarriers = ...
    assign_single_subcarrier_correlation_extrema_to_subcarriers( ...
    max_idx, min_idx, subcarrier_indices);

remove_last_subcarrier = false;

for k = 1 : N_subcarriers

    extrema_in_subcarrier = extrema_in_subcarriers{k};
    maxima_in_subcarrier = extrema_in_subcarrier{1};
    minima_in_subcarrier = extrema_in_subcarrier{2};

    if low_amplitude_subcarriers(k) == 0

        % Previous subcarrier state is only used for subcarrier_number >=
        % 2. For subcarrier number == 1 pass a dummy value.
        if k == 1
            previous_subcarrier_state = -1;
        else
            previous_subcarrier_state = subcarrier_states(k - 1);
        end

        [current_subcarrier_state, return_value] = ...
          determine_state_of_single_subcarrier(...
            maxima_in_subcarrier, minima_in_subcarrier, ...
            correlator_subcarrier_indices, k, previous_subcarrier_state);

        % Error handling
        if return_value ~= 0
            % Case where last subcarrier should be removed
            if return_value == -3
                remove_last_subcarrier = true;
            else
                warning(['Error when determining subcarrier states.', ...
```

```
                    '\nTerminating script.'])
                state_determination_error = true;
            end
        end


        subcarrier_states(k) = current_subcarrier_state;
    end
end
end

function [subcarrier_state, return_value] = ...
    determine_state_of_single_subcarrier(...
    maxima_in_subcarrier, minima_in_subcarrier, subcarrier_indices, ...
    subcarrier_number, previous_subcarrier_state)

N_subcarriers = length(subcarrier_indices) - 1;
subcarrier_start = subcarrier_indices(subcarrier_number);
subcarrier_end = subcarrier_indices(subcarrier_number + 1);

N_max = length(maxima_in_subcarrier);
N_min = length(minima_in_subcarrier);
N_extrema = N_max + N_min;

return_value = 0;
% 0: Success
% -1: Subcarrier frequency or subcarrier duty cycle error (Maximum and
%       minimum in one half)
% -2: Two extrema of the same type detected in current subcarrier
% -3: No local extrema in last subcarrier detected. Remove this
%     subcarrier from response


subcarrier_center = mean([subcarrier_start, subcarrier_end]);

if N_extrema >= 3 || N_extrema == 0

    if N_extrema == 3
        warning(['Possible subcarrier frequency error detected.\n', ...
            'Subcarrier %d contains %d local extrema. This might ', ...
            'denote a subcarrier frequency higher than the nominal ', ...
            'value.\nHave you selected the correct bitrate?' ],...
            subcarrier_number, N_extrema)
        return_value = -1;

    % The distance between the start and the end of the PICC response is
    % ceiled to an integer number of subcarriers.
    % This might result in some cases to an additionally detected
    % subcarrier at the end of the response.
    % This additional subcarrier will be removed from the response.
    %
    % This case is highly relevant when the subcarrier grid is shifted
    % when analyzing a Type B response, because after the shifting
    % procedure a subcarrier is added at the end of the response.
    elseif N_extrema == 0 && subcarrier_number == N_subcarriers
        fprintf(['\tNo local extrema in last subcarrier detected. ', ...
            'Removing this subcarrier from response.'])
        return_value = -3;

    else
        warning(['Possible subcarrier frequency error detected.\n', ...
            'Subcarrier %d does not contain any local extrema. ', ...
            'This might denote a subcarrier frequency significantly ', ...
            'lower than the nominal value. \nHave you selected the ', ...
            'correct bitrate?'], ...
            subcarrier_number)
        return_value = -1;
    end


    subcarrier_state = -1;
```

```
    return
end

% When no phase transition occurs, every subcarrier contains two local
% extrema. One local minimum and one local maximum.
%
% When the maximum is located in the first half and the minimum in the
% second, the subcarrier has a state of 0.
% When the maximum is located in the second half and the minimum in the
% first, the subcarrier has a state of 1.
%
% The case where two extrema are located in the same half denotes a
% possible subcarrier frequency or subcarrier duty cycle error
if N_max == 1 && N_min == 1
    if maxima_in_subcarrier < subcarrier_center && ...
       minima_in_subcarrier > subcarrier_center

       subcarrier_state = 0;

    elseif maxima_in_subcarrier > subcarrier_center && ...
            minima_in_subcarrier < subcarrier_center

        subcarrier_state = 1;

    else
        warning(['Possible subcarrier frequency or subcarrier dury ', ...
            'cycle error detected.\nSubcarrier %d contains one local ',...
            'maximum and one local minimum, but both of them are ', ...
            '\nlocated in one half of the subcarrier.'], ...
            subcarrier_number)
        subcarrier_state = -1;
        return_value = -1;
    end

% Phase transitions are denoted by only one extrema in a subcarrier,
% except for the first subcarrier.
% But only one extrema in the first subcarrier can also be accepted in
% some cases.
elseif N_max == 1 && N_min == 0

    if maxima_in_subcarrier > subcarrier_center
        % Valid phase transition from 0 to 1: Only one extrema in second
        % half of subcarrier and last subcarrier has state 0 or -1.
        % This case is not possible in the first subcarrier.
        if ismember( previous_subcarrier_state, [0,-1] ) && ...
           subcarrier_number > 1

            subcarrier_state = 1;


       % As the first extrema of the correlation signal has only half
       % amplitude, in some cases this extrema is not detected.
       % So the case where the first subcarrier hast only one maximum in
       % its second half results in a subcarrier state of 1
        elseif subcarrier_number == 1
            subcarrier_state = 1;

        % Detected phase transition from 0 to 1 but last subcarrier has
        % state 1 --> bit level encoding error
        else
            warning(['A bit level encoding error was detected. \n', ...
                'Subcarrier %d contains only one local maximum in ', ...
                'its second half which corresponds to a subcarrier ', ...
                'state transition from subcarrier state 0 to ', ...
                'subcarrier state 1.\nBut the previous subcarrier ', ...
                'has also state 1.'], subcarrier_number)
            subcarrier_state = -1;
            return_value = -1;

        end
```

```
    % The case where only one local extremum is contained in the first
    % half of the subcarrier denotes a subcarrier frequency error where
    % the subcarrier frequency is lower than the nominal value, except
    % for the last subcarrier where this case is equal to the case of one
    % detected subcarrier in the second half of the first subcarrier.
    % The case of one local maximum in the first half of the last
    % subcarrier leads to a subcarrier state of 0
    else

        % Current subcarrier is last subcarrier, remove it from the
        % response
        if subcarrier_number == N_subcarriers

            subcarrier_state = 0;

        % When the subcarrier frequency is much lower than its nominal
        % value, the local extrema of the second half of the previous
        % subcarrier moves to the first half of the current subcarrier.
        % This case is handled here.
        else
            subcarrier_state = -1;
            return_value = -1;

            waring(['Possible subcarrier frequency error detected. \n',...
                'Subcarrier %d contains only one local maximum in ', ...
                'first half. This denotes a subcarrier frequency ', ...
                'lower than the nominal value.'], subcarrier_number)

        end


    end

% Handling the same cases as above but for only one minimum contained in
% the subcarrier
elseif N_max == 0 && N_min == 1

    if minima_in_subcarrier > subcarrier_center
        if ismember( previous_subcarrier_state, [1,-1] ) && ...
           subcarrier_number > 1

            subcarrier_state = 0;

        elseif subcarrier_number == 1
            subcarrier_state = 0;

        else
            warning(['A bit level encoding error was detected. \n', ...
                'Subcarrier %d contains only one local minimum in ', ...
                'its second half which corresponds to a subcarrier ', ...
                'state transition from subcarrier state 1 to ', ...
                'subcarrier state 0.\nBut the previous subcarrier ', ...
                'has also state 0.'], subcarrier_number)
            subcarrier_state = -1;
            return_value = -1;
        end
    else
        if subcarrier_number == N_subcarriers

            subcarrier_state = 1;

        else
            subcarrier_state = -1;
            return_value = -1;

            waring(['Possible subcarrier frequency error detected. \n',...
                'Subcarrier %d contains only one local minimum in ', ...
                'first half. This denotes a subcarrier frequency ', ...
                'lower than the nominal value.'], subcarrier_number)
        end
```

```
    end

% Error case: Two extrema in one subcarrier, but both of them are maxima
% or minima
elseif N_max == 2 || N_min == 2
    if N_max == 2
        warning('Subcarrier %d contains 2 extrema but both of them ', ...
        'are maxima.', k)
    else
        warning('Subcarrier %d contains 2 extrema but both of them ', ...
        'are minima.', k)
    end

    subcarrier_state = -1;
    return_value = -2;
end


end

function extrema_in_subcarriers = ...
    assign_single_subcarrier_correlation_extrema_to_subcarriers( ...
    max_idx, min_idx, subcarrier_indices)

% function extrema_in_subcarriers = ...
%    assign_single_subcarrier_correlation_extrema_to_subcarriers( ...
%    max_idx, min_idx, subcarrier_indices)
%
% DESCRIPTION: Function for assigning the local extrema from the single
%              subcarrier correlation signal to their corresponding
%              subcarriers
%
%
% INPUT:
%   max_idx.....Indices of the local maxima
%   min_idx.....Indices of the local minima
%   subcarrier_indices.....Vector of subcarrier boundaries
%
%
% OUTPUT:
%   extrema_in_subcarriers.....number_of_subcarriers x 1 cell array
%       containing in each element a 1x2 cell array whose first element
%       contains the vector of local maxima in the corresponsing
%       subcarrier. The second element conatins the vector of local minima
%       in the subcarrier.
%
%       A qualitative sketch of the cell array's structure is depicted
%       below:
%
%       |{}| --> |{[indices of local maxima],[indices of local minima]}|
%       |{}|
%        .
%        .
%        .
%       |{}|
%

N_subcarriers = length(subcarrier_indices) - 1;
samples_per_subcarrier = subcarrier_indices(2) - subcarrier_indices(1);

extrema_in_subcarriers = cell(N_subcarriers, 1);

% Align baseband subcarrier indices to correlation signal
correlator_subcarrier_indices = subcarrier_indices - ...
    round( 3/4 * samples_per_subcarrier );

for k = 1 : N_subcarriers
    subcarrier_start = correlator_subcarrier_indices(k);
    subcarrier_end = correlator_subcarrier_indices(k + 1);
```

```
    maxima_in_subcarriers = max_idx( (max_idx > subcarrier_start) & ...
        (max_idx < subcarrier_end) );

    minima_in_subcarriers = min_idx( (min_idx > subcarrier_start) & ...
        (min_idx < subcarrier_end) );

    extrema_in_subcarriers{k} = {maxima_in_subcarriers, ...
      minima_in_subcarriers};
end


end

% ========== Dual subcarrier correlation processing funcitons ===========

function xc = phase_transition_correlation(baseband_channel, ...
    samples_per_subcarrier)

% function xc = phase_transition_correlation(baseband_channel, ...
%   samples_per_subcarrier)
%
%
% DESCRIPTION: Function for performing the subcarrier phase transition
%              cross-correlation.
%              The cross convolution is used instead of the
%              cross-correlation to automatically remove the mean from the
%              signal to correlate
%
%
% INPUT:
%   baseband_channel.....Baseband channel to correlate
%   samples_per_subcarrier.....Samples in one subcarrier
%
%
% OUTPUT:
%   xc.....Resulting crorr-correlation signal
%

% ----- Add backoff to start and end of signal
N_backoff = 2 * samples_per_subcarrier;

modified_baseband_channel = [baseband_channel(1) * ones(N_backoff, 1); ...
    baseband_channel;
    baseband_channel(end) * ones(N_backoff, 1)];

% ----- Wavelet generation
%
% Baseband channel is correlated with a subcarrier phase transition
% sequence consisting of one in-phase and one inverse-phase subcarrier

samples_in_half_subcarrier = samples_per_subcarrier / 2;

subcarrier = [ones(samples_in_half_subcarrier, 1); ...
    -ones(samples_in_half_subcarrier, 1)];
wavelet = [subcarrier ; flipud(subcarrier)];

xc = xcov(modified_baseband_channel, wavelet);
N_sig = length(modified_baseband_channel);

%Select upper half of correlation signal and remove backoff
xc = xc(N_sig + N_backoff  : end - N_backoff);

end


function [max_idx, max_vals, min_idx, min_vals] = ...
    determine_phase_transition_correlation_extrema(xc, ...
    subcarrier_indices, env_up, env_lo, encoding)

% function [max_idx, max_vals, min_idx, min_vals] = ...
%    determine_phase_transition_correlation_extrema(xc, ...
```

```
%    subcarrier_indices, env_up, env_lo)
%
%
%
% DESCRIPTION: Function for determining the local extrema from the
%              phase-transition correlation signal
%
%
% INPUT:
%   xc.....Phase-transition cross-correlation signal
%   subcarrier_indices.....Vector of subcarrier boundaries
%   env_up.....Upper peak envelope from single subcarrier correlation
%               signal
%   env_lo.....Lower envelope from single subcarrier correlation signal
%   encoding.....Structure containg the specified encoding of the
%                response
%
%
% OUTPUT:
%   max_idx.....Vector of indices of local maxima in correlation signal
%   max_vals.....Vector of values of local minima in correlation signal
%   min_idx.....Vector of indices of local maxima in correlation signal
%   min_vals.....Vector of values of local minima in correlation signal
%

% Enable plot functionality
PLOT = 0;

samples_per_subcarrier = subcarrier_indices(2) - subcarrier_indices(1);

% When correlating with a subcarrier phase transition sequence, the
% resulting correlation signal is zero in areas with no phase transition.
% (Except fade-in and fade-out effects)
%
% In case of a phase transition 3 local extrema occur with a distance of
% half of a subcarrier each:
% 1.) Local extrema with half height
% 2.) Local extrema of inverse polarity with full height
% 3.) Local extrema with half height
%
% The algorithm wants to detect the second extrema mentioned above. Those
% points determine a point of maximum correlation between the wavelet and
% the baseband channel and, due to the choice of the wavelet as a phase
% transition, determine a subcarrier phase transition.
%
% When assigning the subcarrier grid to the resulting correlation signal,
% one can see that e.g. if the subcarrier phase transition occurs between
% the 3rd and the 4th subcarrier, the peak to detect is located at the
% left boundary (the start) of the 3rd subcarrier. Therefore the
% subcarrier grid is shifted back by 3/2 subcarries to simplify
% peak-to-phase-transition assignment as the peak to detect would now
% occur in the center of the 4th subcarrier.
%
% This convention, that in case of a valid subcarrier phase transition,
% the peaks from the correlation signal are located in the center of the
% first subcarrier with the 'new' phase, will be used for error-detection
% and plausibility checking.

correlator_subcarrier_indices = subcarrier_indices - 3/2 * ...
    samples_per_subcarrier;


% ----- Dynamic threshold calculation
%
% Compute a dynamic threshold from the upper and lower peak envelopes
% from single subcarrier correlation.
% A dynamic threshold is needed because the amplitudes of the peaks to
% detect (see below) can vary in a large range at large interstate phase
% drifts.
threshold = abs(env_up - env_lo);
```

```
% Exclude high level areas in envelopes from calculation of normalization
% factor
idx = subcarrier_indices(2) : subcarrier_indices(end -1);
threshold = threshold / abs(max(env_up(idx)) - min(env_lo(idx)));

% The peak envelopes used as dynamic thresholds for peak detection are
% calculated from the single subcarrier correlation signal. Due to the
% different correlator wavelets the threshold must be aligned to the
% current correlation signal for proper peak detection.

% Empirically determined index alignment.
threshold_alignment = 1/2 * samples_per_subcarrier;

threshold = [threshold(threshold_alignment : end) ;
    threshold(end) * ones( threshold_alignment - 1, 1)];

% To avoid invalid extrema due to fade-in and fade-out effects at start
% and end of the correlation signal the normalized threshold is set to
% 1.1 at the first and last 2 subcarriers (empirically determined).
% Only modify the threshold at the last 2 subcarriers in case of bitrates
% lower than 848 kBps because for bitrates >= 848 kBps there is then only
% one subcarrier contained in each etu. Therefore a subcarrier
% phase transition can also occur in the last or last but one subcarrier.

threshold(1 : correlator_subcarrier_indices(3)) = 1.1;

bitrate = str2double(encoding.bitrate);
if bitrate < 848
    threshold(correlator_subcarrier_indices(end - 2) : end) = 1.1;
end


% ----- Detect and filter extrema

[max_vals, max_idx, min_vals, min_idx] = ...
    find_and_cleanup_peaks(xc, correlator_subcarrier_indices(1), ...
    correlator_subcarrier_indices(end));

% Do not use absolute maximum or minimum as reference for threshold but
% calculate mean value of peaks around those values to reduce the impact
% of single, high amplitude extrema
abs_max = max(max_vals);
abs_min = min(min_vals);

vals_for_mean_max = max_vals(max_vals > 0.95 * abs_max);
vals_for_mean_min = min_vals(min_vals < 0.95 * abs_min);

mean_max = mean(vals_for_mean_max);
mean_min = mean(vals_for_mean_min);

% Empirically determined scaling factor for proper peak detection
threshold_scaling = 0.95;


max_idx_ = max_idx;
max_idx(max_vals < threshold_scaling * mean_max * threshold(max_idx_))...
  = [];
max_vals(max_vals < threshold_scaling * mean_max * threshold(max_idx_))...
  = [];

min_idx_ = min_idx;
min_idx(min_vals > threshold_scaling * mean_min * threshold(min_idx_))...
  =[];
min_vals(min_vals > threshold_scaling * mean_min * threshold(min_idx_))...
  =[];



if PLOT
    figure()
    l1 = plot(xc);
```

```
    hold on

    height = max(xc) + 0.1 * abs(max(xc) - min(xc));
    l2 = line(correlator_subcarrier_indices * [1,1], ...
       [1.15 * height, min(xc)], 'Color', [0,0,0]);
    l3 = plot(max_idx, max_vals, 'ro');
    l4 = plot(min_idx, min_vals, 'go');
    l5 = plot(threshold_scaling * mean_max * threshold);
    l6 = plot(threshold_scaling * mean_min * threshold);

    height = max(xc) + 0.15 * abs(max(xc) - min(xc));
    for k = 1 : length(subcarrier_indices) - 1
       subcarrier_start = correlator_subcarrier_indices(k);
       subcarrier_end = correlator_subcarrier_indices(k + 1);

       text(round(mean([subcarrier_start, subcarrier_end])), height, ...
           num2str(k), 'HorizontalAlignment', 'center')
    end
    grid minor
    title('Correlation signal for phase transition detection.')
    legend([l1, l2(end), l3, l4, l5, l6], {'Correlation signal', ...
       'Subcarrier boundaries', 'phase transition', ...
       'phase transition', 'Threshold for maxima', ...
       'Threshold for minima'})

end


end


function phase_transition_data = ...
  assign_phase_transition_extrema_to_subcarriers(max_idx, min_idx, ...
  subcarrier_indices)

% function phase_transition_data = ...
%  assign_phase_transition_extrema_to_subcarriers(max_idx, min_idx, ...
%  subcarrier_indices)
%
%
%
% DESCRIPTION: Function for assigning the extrema determined by
%              determine_phase_transition_correlation_extrema() to the
%              corresponding subcarriers
%
%
% INPUT:
%   max_idx.....Indices of local maxima in correlation signal
%   min_idx.....Indices of local minima in correlation signal
%   subcarrier_indices.....Vector of baseband aligned subcarrier indices
%
%
% OUTPUT:
%   phase_transition_data.....Matrix containing the local extrema assigned
%       to the corresponding subcarrier.
%       Each row of the matrix is of the following form:
%
%       [number of subcarrier, global index of extrema, extrema type]
%
%       -) number of subcarrier: Integer number of the subcarrier assigned
%           to the local extrema
%       -) global index of the extrema: Entry of max_idx or min_idx
%           assigned to the subcarrier in column 1
%       -) extrema type: Integer number specifying the type of the extrema
%           in the subcarrier.
%               .) 1 : local maxima
%               .) 0 : local minima
%


samples_per_subcarrier = subcarrier_indices(2) - subcarrier_indices(1);
```

```
correlator_subcarrier_indices = subcarrier_indices - 3/2 * ...
  samples_per_subcarrier;

phase_transition_data_max = [ceil((max_idx - ...
    correlator_subcarrier_indices(1)) / samples_per_subcarrier), ...
    max_idx, ones(length(max_idx), 1)];

phase_transition_data_min = [ceil((min_idx - ...
    correlator_subcarrier_indices(1)) / samples_per_subcarrier), ...
    min_idx, zeros(length(min_idx), 1)];

phase_transition_data = [phase_transition_data_max; ...
  phase_transition_data_min];
[phase_transition_data, ~] = sortrows(phase_transition_data, 1);

end


function check_passed = phase_transition_distance_check(...
    max_idx, min_idx, samples_per_subcarrier)

% function check_passed = phase_transition_distance_check(...
%    max_idx, min_idx, samples_per_subcarrier)
%
% DESCRIPTION: Part 1 of the phase transition distance plausibility check
%
%              This function checks if all distances between two
%              consecutive phase jumps are an integer number of
%              subcarriers.
%              Due to none ideal signal behaviour a maximum deviation of
%              1/8 subcarrier is allowed.
%
%
% INPUT:
%   max_idx.....Index vector of local maxima
%   min_idx.....Index vector of local minima
%   samples_per_subcarrier.....Samples in one subcarrier
%
%
% OUTPUT:
%   check_passed.....Binary value (true/false) indicating if the check
%       was passed or not
%


fprintf('Phase transition distance check:\n')
[extrema_idx, ~] = sortrows([max_idx; min_idx]);

check_passed = true;
N_phase_transitions = length(extrema_idx);

% Empirically determined deviation limit
scaler = 1/8;
distance_deviation_limit = round(scaler * samples_per_subcarrier);

for k = 1 : N_phase_transitions - 1
    current_transition_peak = extrema_idx(k);
    next_transition_peak = extrema_idx(k + 1);

    distance = next_transition_peak - current_transition_peak;

    % Nearest integer number of subcarriers between extrema
    n = round(distance / samples_per_subcarrier);

    % Deviation from integer number of subcarriers
    deviation = distance - n * samples_per_subcarrier;

    if abs(deviation) > distance_deviation_limit
        warning(['\tDistance between phase transition peak ', ...
            'at index %d and phase transition peak at index %d ', ...
            'is not an integer number of subcarrier periods.\n\t', ...
```

```
            'Maximum allowed deviation from integer number of ', ...
            'subcarriers is %d samples (%.3f * subcarrier). ', ...
            'Measured deviation is %d samples. One subcarrier ', ...
            'contains %d samples.'], ...
            current_transition_peak, next_transition_peak, ...
            distance_deviation_limit, scaler, deviation, ...
            samples_per_subcarrier)

        check_passed = false;
    end
end

if check_passed
    fprintf('\tPassed\n\r');
end

end

function check_passed = phase_transition_sequence_check(max_idx, ...
    min_idx, low_amplitude_subcarriers, subcarrier_indices)

% function check_passed = phase_transition_sequence_check(max_idx, ...
%    min_idx, low_amplitude_subcarriers, subcarrier_indices)
%
% DESCRIPTION: Function for checking if every local maxima denoting a
%   phase transition is followed by a local minima denoting an inverse
%   phase transition
%
%
% INPUT:
%   max_idx.....Index vector of local maxima
%   min_idx.....Index vector of local minima
%   subcarrier_indices.....Vector of subcarrier boundaries
%   low_amplitude_subcarriers.....Binary vector indicating subcarriers
%       with low amplitudes
%
%
% OUTPUT:
%   check_passed.....Binary value (true/false) indicating if the check
%       was passed or not
%

fprintf('Phase transition sequence check:\n')

check_passed = true;
samples_per_subcarrier = subcarrier_indices(2) - subcarrier_indices(1);

% Align baseband channel subcarrier indices to correlation signal
correlator_subcarrier_indices = subcarrier_indices - 3/2 * ...
  samples_per_subcarrier;

% Create a matrix used for further processing, containing the index and
% type of each extrema in each row. The rows are sorted ascending.
extrema_data = [ max_idx, ones(length(max_idx), 1);
                 min_idx, zeros(length(min_idx), 1)];
[extrema_data, ~] = sortrows(extrema_data, 1);

N_phase_transitions = length(extrema_data);

for k = 1 : N_phase_transitions - 1

    current_phase_transition_type = extrema_data(k, 2);
    next_phase_transition_type = extrema_data(k + 1, 2);

    % The type of the current extrema has to be different from the type
    % of the next extrema.
    % Two consecutive extrema of the same type correspond to two
    % consecutive phase transitions of the same type.
    if current_phase_transition_type == next_phase_transition_type

        % Assign extrema to subcarriers to determine if there are
```

```
        % subcarriers with low amplitude between them.
        % If there are low amplitude subcarriers between them no error
        % occurred.
        current_phase_transition_idx = extrema_data(k, 1);
        next_phase_transition_idx = extrema_data(k + 1, 1);

        subcarrier_of_current_extremum = ceil((...
            current_phase_transition_idx - ...
            correlator_subcarrier_indices(1)) / samples_per_subcarrier);

        subcarrier_of_next_extremum = ceil((...
            next_phase_transition_idx - ...
            correlator_subcarrier_indices(1)) / samples_per_subcarrier);

        idx = subcarrier_of_current_extremum : ...
          subcarrier_of_next_extremum;
        low_amplitude_subcarriers_between_peaks = ...
          low_amplitude_subcarriers(idx);

        % As mentioned above only of no low amplitude subcarriers are
        % located between the two phase transitions, an error occurred.
        if ~any(low_amplitude_subcarriers_between_peaks)

            if current_phase_transition_type && ...
              ~next_phase_transition_type
                str1 = 'maximum';
                str2 = 'minimum';
            else
                str1 = 'minimum';
                str2 = 'maximum';
            end

            warning(['Phase transition peak at index %d is supposed ', ...
                'to be a local %s because previous phase transition ', ...
                'at index %d is a local %s. \nTherefore two ', ...
                'consecutive phase transitions of the same type ', ...
                'were detected.'], ...
                current_phase_transition_idx, str2, ...
                next_phase_transition_idx, str1)

            check_passed = false;
        end
    end

end

if check_passed
    fprintf('\tPassed\n\r')
end
end

function check_passed = nominal_phase_transition_position_check(...
    phase_transition_data, subcarrier_indices)

fprintf('Nominal phase transition position check:\n')
check_passed = true;

samples_per_subcarrier = subcarrier_indices(2) - subcarrier_indices(1);
correlator_subcarrier_indices = subcarrier_indices - 3/2 * ...
  samples_per_subcarrier;

distances_to_subcarrier_centers = ...
    calculate_distances_to_subcarrier_centers(...
    phase_transition_data, correlator_subcarrier_indices);

max_nominal_position_deviation = ...
  max(abs(distances_to_subcarrier_centers));

deviation_factor = 0.1;
phase_transition_deviation_limit = ...
  round(deviation_factor * samples_per_subcarrier);
```

```
% Maximum deviation in percent for printing to console
tmp = max_nominal_position_deviation / samples_per_subcarrier * 100;
fprintf(['\tMaximum deviation of phase transitions to nominal ', ...
    'positions: %4.2f%% (%d samples) of one subcarrier period ', ...
    '(%d samples).\n\t', ...
    'Limit is %4.2f%% (%d samples) of one subcarrier period ', ...
    '(%d samples).\n\r'], ...
    tmp, round(max_nominal_position_deviation), samples_per_subcarrier,...
    deviation_factor*100, round(phase_transition_deviation_limit), ...
    samples_per_subcarrier)

nominal_position = max_nominal_position_deviation < ...
    phase_transition_deviation_limit;

if nominal_position

    fprintf('\tPhase transitions at nominal position.\n\r')
else

    warning(['Phase transitions not at nominal position. ', ...
        'Terminating script.'])
    check_passed = false;
end
end

function subcarrier_states = ...
    determine_subcarrier_states_from_phase_transitions( ...
    phase_transition_data_re, phase_transition_data_im, ...
    initial_subcarrier_state, number_of_subcarriers)


subcarrier_states = -ones(number_of_subcarriers, 1);

% ----- Merge phase transitions from both channels of baseband signal
[tmp_phase_transitions, ~] = sortrows([phase_transition_data_re(:, 1);
                              phase_transition_data_im(:, 1)]);
phase_transitions = [];
for k = 2 : length(tmp_phase_transitions)

    if tmp_phase_transitions(k) ~= tmp_phase_transitions(k-1)
        phase_transitions = [phase_transitions; ...
            tmp_phase_transitions(k - 1)];
    end

end

% Algorithm above does not add last phase transition.
phase_transitions = [phase_transitions; tmp_phase_transitions(end)];

% ----- Determine subcarrier states using phase transitions
current_subcarrier_state = initial_subcarrier_state;
start_idx = 1;
for k = 1 : length(phase_transitions)

    end_idx = phase_transitions(k) - 1;
    subcarrier_states(start_idx : end_idx) = current_subcarrier_state;

    start_idx = end_idx + 1;
    current_subcarrier_state = ~current_subcarrier_state;

end
subcarrier_states(start_idx : end) = current_subcarrier_state;

end

% ==================== Miscellaneous functions ========================

function [max_vals, max_idx, min_vals, min_idx] = ...
    find_and_cleanup_peaks(signal, start_idx, end_idx)
```

```
% function [max_vals, max_idx, min_vals, min_idx] = ...
%    find_and_cleanup_peaks(subcarrier_indices, correlation_signal)
%
% DESCRIPTION: Function for finding local maxima and minima in the
% correlation signal and removing unnecessary ones
%
%
% INPUT:
%        signal.......The correlation signal whose peaks should
%                              be determined
%        start_idx.................Extrema before this index are removed
%        end_idx...................Extrema after this index are removed
%
% OUTPUT:
%       max_vals.....The values of the found local maxima
%       max_idx......The indices of the found local maxima
%       min_vals.....The values of the found local minima
%       min_idx......The indices of the found local minima


[max_vals, max_idx] = new_findpeaks(signal);

[min_vals, min_idx] = new_findpeaks(-signal);
min_vals = -min_vals;

max_vals(max_idx < start_idx) = [];
max_idx(max_idx < start_idx) = [];

min_vals(min_idx < start_idx) = [];
min_idx(min_idx < start_idx) = [];


max_vals(max_idx > end_idx) = [];
max_idx(max_idx > end_idx) = [];

min_vals(min_idx > end_idx) = [];
min_idx(min_idx > end_idx) = [];

end

function [shift_subcarriers, error] = Type_B_subcarrier_shift(max_idx, ...
    min_idx, subcarrier_indices, encoding)

% function [shift_subcarriers, error] = Type_B_subcarrier_shift(max_idx,
%    min_idx, subcarrier_indices, encoding)
%
% DESCRIPTION: Function for determining if a subcarrier shift for Type B
%              PICC responses is required.
%
% When analysing an unipolar modulated Type B response the following case
% might occur:
%
% The start-and-end-detection algorithm detects the first edge of the
% baseband signal.
% If the first subcarrier in the response starts with MS2
% (subcarrier state 0), and MS2 is almost equal or equal to US, the
% calculated subcarrier grid is delayed by half of a subcarrier because
% the edge in the transition from MS2 to MS1 is detected as start of
% response.
%
% In this case the subcarrier grid has to be shifted by half of a
% subcarrier. To determine this case the peaks detected by
% phase-transition correlation are analyzed.
% In case that no subcarrier shift is required, the peaks will be located
% around the centers of subcarrier in the correlation-signal aligned
% subcarrier grid
%
% In case that a shift is required those peaks are located around the
% subcarrier boundaries.
%
% A shift denotes that the response starts with state MS2,
```

```
% (= subcarrier state 0) otherwise it starts with MS1
% (= subcarrier state 1)
%
%
% INPUT:
%   max_idx.....Index vector of local maxima
%   min_idx.....Index vector of local minima
%   subcarrier_indices.....Vector of subcarrier boundaries
%   encoding.....Structure containing the specified encoding of the signal
%
%
% OUTPUT:
%   shift_subcarriers.....Binary value indicating if the subcarrier has to
%       be shifted or not
%   error.....Binary value indicating if an error occurred.
%       An error occurred when as many peaks are located at subcarrier
%       boundaries then peaks located at subcarrier centers.
%       In this case, no clear result can be determined whether a
%       subcarrier shift is required or not.
%

fprintf('Type B PICC response subcarrier shift:\n')
error = false;

samples_per_subcarrier = subcarrier_indices(2) - subcarrier_indices(1);
correlator_subcarrier_indices = subcarrier_indices - ...
      3/2 * samples_per_subcarrier;


phase_transition_data = assign_phase_transition_extrema_to_subcarriers(...
    max_idx, min_idx, subcarrier_indices);

distances_to_subcarrier_centers = ...
    calculate_distances_to_subcarrier_centers(phase_transition_data, ...
    correlator_subcarrier_indices);

distances_to_subcarrier_centers = abs(distances_to_subcarrier_centers);

bitrate_int = str2double(encoding.bitrate);

% Empirically determined, bitrate dependent deviation limit
if bitrate_int <= 848
    scaler = 1 / 8;
    center_deviation_limit = round(samples_per_subcarrier  * scaler);
else
    scaler = 1 / 4;
    center_deviation_limit = round(samples_per_subcarrier * scaler);
end

peaks_within_subcarrier_centers = distances_to_subcarrier_centers < ...
    center_deviation_limit;

peaks_within_subcarrier_centers = double(peaks_within_subcarrier_centers);

peaks_within_subcarrier_centers(peaks_within_subcarrier_centers == 0) =...
    -1;

subcarrier_shift_decision_value = sum(peaks_within_subcarrier_centers);
if subcarrier_shift_decision_value < 0
    shift_subcarriers = true;

    fprintf(['\t%d subcarrier periods are located within %.3f ', ...
        'subcarrier periods around their corresponding subcarrier ', ...
        'centers and \n\t%d subcarrier periods are located outside ', ...
        'this limit. \n\t', ...
        'Therefore the subcarrier boundaries are shifted back by ', ...
        'half of a subcarrier period.\n\r'], ...
        sum(peaks_within_subcarrier_centers == 1), scaler, ...
        sum(peaks_within_subcarrier_centers == -1))

elseif subcarrier_shift_decision_value > 0
```

```
    shift_subcarriers = false;

    fprintf(['\t%d subcarrier periods are located within %.3f ', ...
        'subcarrier periods around their corresponding subcarrier ', ...
        'centers and \n\t%d subcarrier periods are located outside ', ...
        'this limit. \n\t', ...
        'Therefore the subcarrier boundaries are NOT shifted back by ',...
        'half of a subcarrier period.\n\r'], ...
        sum(peaks_within_subcarrier_centers == 1), ...
        scaler, sum(peaks_within_subcarrier_centers == -1))
else
    warning(['Evaluation whether a subcarrier shift for the Type B ', ...
        'response is required or not has no clear \nresult because ', ...
        'the same number of phase transition peaks that are located ', ...
        'within %.3f subcarrier periods \naround the corresponding ', ...
        'subcarrier centers are located outside this limit.'], ...
        scaler)
    shift_subcarriers = false;
    error = true;
end

end

function distances_to_subcarrier_centers = ...
    calculate_distances_to_subcarrier_centers(phase_transition_data, ...
    subcarrier_indices)

% function distances_to_subcarrier_centers = ...
%    calculate_distances_to_subcarrier_centers(phase_transition_data, ...
%    subcarrier_indices)
%
% DESCRIPTION: Function for computing the distances of the input local
%              extrema to the centers of their corresponding subcarriers
%
%
% INPUT:
%   phase_transition_data.....Matrix as returned by
%       assign_phase_transition_extrema_to_subcarriers()
%    subcarrier_indices.....Vector of subcarrier boundaries
%
%
% OUTPUT:
%   distances_to_subcarrier_centers.....Vector containing the distances of
%       the local extrema to their corresponding subcarrier centers
%

subcarriers_with_phase_transitions = phase_transition_data(:, 1);

subcarriers_start = subcarrier_indices(...
    subcarriers_with_phase_transitions);
subcarriers_end = subcarrier_indices(...
    subcarriers_with_phase_transitions + 1);

% Compute the mean value of each row to get the subcarrier center.
subcarrier_centers = mean([subcarriers_start, subcarriers_end], 2);

extrema_idx = phase_transition_data(:, 2);
distances_to_subcarrier_centers = subcarrier_centers - extrema_idx;

end

function subcarrier_indices = subcarrier_grid_correction(...
    phase_transition_data, subcarrier_indices)

% function subcarrier_indices = subcarrier_grid_correction(...
%    phase_transition_data, subcarrier_indices)
%
% DESCRIPTION: Move the passed subcarrier boundaries in a way that the
%   phase transition peaks are centered.
%
%   In case of an ideal signal, the peaks detected from the
```

```
%   phase-transition correlation are perfectly centered in each
%   correlation signal aligned subcarrier. Due to none-ideal signals the
%   subcarrier grid determined from start and end of the PICC response
%   might slightly vary from this.
%
%   Therefore the subcarrier grid is moved in a way that the detected
%   peaks are centered or that at least the distances between the peaks
%   and the subcarrier centers are evenly distributed over the whole
%   response.
%
%
% INPUT:
%   phase_transition_data.....Matrix as returned by
%       assign_phase_transition_extrema_to_subcarriers()
%    subcarrier_indices.....Vector of subcarrier boundaries
%
%
% OUTPUT:
%   subcarrier_indices.....Modified subcarrier indices
%

samples_per_subcarrier = subcarrier_indices(2) - subcarrier_indices(1);
correlator_subcarrier_indices = subcarrier_indices - ...
    3/2 * samples_per_subcarrier;

distances_to_subcarrier_centers = ...
    calculate_distances_to_subcarrier_centers(...
    phase_transition_data, correlator_subcarrier_indices);

% Use the 2 times the standard deviation as threshold for ignoring outlied
% peaks
threshold =  2 * sqrt(var(distances_to_subcarrier_centers));
mean_distance_to_centers = mean(distances_to_subcarrier_centers);

% Do not use phase transitions with absolute distances to their centers of
% more than 2 time the mean value of all distances

idx = abs(distances_to_subcarrier_centers) > 2 * ...
    abs(mean_distance_to_centers) + threshold;
distances_to_subcarrier_centers(idx) = [];


% The final subcarrier grid correction value is the mean value of all
% distances without outliers
subcarrier_correction = round(mean(distances_to_subcarrier_centers));

% The distances are calculated as center_idx - extrema_idx. Therefore, if
% the mean distance is positive, the subcarrier boundaries has to be
% shifted back.
subcarrier_indices = subcarrier_indices - subcarrier_correction;
end

function [check_passed, remove_last_subcarrier] = ...
    phase_transition_comparision_check(phase_transition_data, ...
    subcarrier_states)

% function [check_passed, remove_last_subcarrier] = ...
%    phase_transition_comparision_check(phase_transition_data, ...
%    subcarrier_states)
%
% DESCRIPTION: Function for checking if the phase transitions detected by
%   the phase transition correlation are the same as the phase transitions
%   from the single subcarrier correlation.
%
% Used algorithm + example:
%-----------------------------
% -) subcarrier_states = 0 0 0 1 1 0 -1 0 0 1 1 -1 -1
% -) Setting all '-1' to '3': 0 0 0 1 1 0  3 0 0 1 1  3  3
%       The phase transition detection is done by detecting the
%       subcarrier indices where the subcarrier states change.
%       Low amplitude subcarriers have a state of -1. To avoid phase
```

```
%       transition detection on transitions from state 0 or 1 to -1,
%       all -1 are set to 3
% -) Differentiate the vector from the previous step
%       --> d = 0 0 -1 0 -3 0 0 -1 0 -2 0
% -) Set +-3 and +-2 to 0. Now the result from the previous step only
%    contains non-zero elements at the last subcarriers in consecutive
%    sequences with the same subcarrier states
%    --> d = 0 0 -1 0 0 0 0 -1 0 0 0
% -) The first column of phase_transition_data contains the indices of
%    first subcarriers with new state in a sequence. Therefore add a zero
%    at the begin of the result from the previous step
%    --> d = [0;d] = 0 0 0 -1 0 1 0 0 0 -1 0 0 0
% -) Find indices of non-zero elements. --> phase_transitions = [4,6,10]
% -) Check if an equal number of phase transitions were found
%       a) If more were found from single subcarier correlation, check not
%           passed.
%       b) If more were found from phase transition correlation, ignore
%           the missing ones that occur right after a sequence of low
%           amplitude subcarriers by removing them from
%           phase_transition_data
%           (for further explanation on this case see the code below)
%           Continue with the next step using the modified variables
%
% -) If an equal number of phase transitions were found check if the phase
%    transitions occur at the same position.
%       .) If yes, check passed
%       .) If not, check failed
%
%
% INPUT:
%   phase_transition_data.....Matrix returned from
%       assign_phase_transition_extrema_to_subcarriers() containing
%       information about the subcarrier phase-transitions detected by the
%       phase-transition correlation
%    subcarrier_states.....Vector containing the subcarrier states
%       (0, 1, -1) from the single subcarrier correlation
%
%
% OUTPUT:
%   check_passed.....Binary value indicating whether the check was passed
%       or not.
%   remove_last_subcarrier.....Binary value indicating whether the last
%       subcarrier in the response can be removed or not.

    check_passed = true;
    remove_last_subcarrier = false;

    [N_phase_transitions, ~] = size(phase_transition_data(:, 1));
    N_subcarriers = length(subcarrier_states);

    tmp_subcarrier_states = subcarrier_states;
    tmp_phase_transition_data = phase_transition_data;

    tmp_subcarrier_states(subcarrier_states == -1) = 3;
    d = diff(tmp_subcarrier_states);
    d(ismember(abs(d), [2, 3])) = 0;
    d = [0; d];
    phase_transitions = find(d ~= 0);

    % Tests showed that the last transition from MS1/MS2 to US might be
    % detected as a subcarrier with one local extrema which might be
    % interpreted as phase transition.
    % A phase transition in the last subcarrier is valid, but only for
    % Type B signals (in other cases the decoder will throw an error).
    %
    % This if-clause handles the case where a phase transition in the last
    % subcarrier is detected by single subcarrier correlation but not from
    % phase transition correlation.
    % In this case the last subcarrier can be removed from the response.
    if phase_transitions(end) == N_subcarriers && ...
            ~ismember(phase_transitions(end), phase_transition_data(:, 1))
```

```
        remove_last_subcarrier = true;
        phase_transitions = phase_transitions(1 : end - 1);
    end


    if length(phase_transitions) > N_phase_transitions
            warning(['More phase transitions detected from single ', ...
                'subcarrier correlation than from\nphase-transition ', ...
                'correlation.\nPhase transition positions detected ', ...
                'from single subcarrier correlation:\n\t%s\nPhase ', ...
                'transitions detected from phase-transition ', ...
                'correlation:\n\t%s\n '], ...
                sprintf('%d ', phase_transitions), ...
                sprintf('%d ', phase_transition_data(:, 1)))

            check_passed = false;

    % Imagine the following case:
    % subcarrier_states = 0 0 1 -1 -1 1 0 0 1
    % --> phase transitions = [3, 9]
    % but phase_transition_data(:, 1) = [3, 6, 9]
    %
    % The phase transition from subcarrier 5 to subcarrier 6 cannot be
    % determined from the single subcarrier correlation signal due to the
    % low amplitude subcarrier sequence before.
    % This case always occurs when a phase transition occurs at the first
    % subcarrier after a low amplitude subcarrier period. (Replace the -1
    % from above by 0)
    %
    % In this case, the missing phase transitions that meet the conditions
    % from above are ignored and no error is throw.
    % In the example, the missing phase transition at subcarrier 6 is
    % ignored.
    elseif length(phase_transitions) < N_phase_transitions

        [check_passed, missing_phase_transition_idx] = ...
            phase_transition_comparision_check_step_b(...
            phase_transitions, phase_transition_data, subcarrier_states);

        if check_passed
            tmp_phase_transition_data(missing_phase_transition_idx, :) ...
                = [];
        % Phase transitions detected from phase_transition_data were
        % missing in phase transitions detected from single subcarrier
        % correlation.
        else
            check_passed = false;
            return
        end

    end


    % tmp_phase_transition_data might be modified
    [N_phase_transitions, ~] = size(tmp_phase_transition_data);

    % Check if both vectors have the same length
    if length(phase_transitions) == N_phase_transitions
        % And if same phase transitions were detected.
        if any(~ismember(phase_transitions, ...
                tmp_phase_transition_data(:, 1)))
            warning(['Unequal phase transitions detected from single ',...
                'subcarrier correlation and\n phase transition ', ...
                'correlation.\nPhase transitions from single ', ...
                'subcarrier correlation:\n\t%s\nPhase transitions ', ...
                'from phase_transition_correlation:\n\t%s\n'], ...
                sprintf('%d ', phase_transitions), ...
                sprintf('%d ', phase_transition_data(:, 1)))

            check_passed = false;
```

```
        end
    % Handle case if after all there is still an additional phase
    % transition detected.
    else
        warning(['Phase transitions detected that occur in single ', ...
            'subcarrier correlation but not\nin-phase transition ', ...
            'correlation.\n', ...
            'Phase transitions from single subcarrier ', ...
            'correlation:\n\t%s\n', ...
            'Phase transitions from phase transition ', ...
            'correlation:\n\t%s\n'], ...
            sprintf('%d ', phase_transitions), ...
            sprintf('%d ', phase_transition_data(:, 1)))

        check_passed = false;
    end


end

function [check_passed, missing_phase_transitions_idx] = ...
    phase_transition_comparision_check_step_b(phase_transitions, ...
    phase_transition_data, subcarrier_states)


check_passed = true;

%See example: missing_phase_transitions_idx = 2
missing_phase_transitions_idx = find( ...
    ~ismember(phase_transition_data(:, 1), phase_transitions));

% See example: phase_transition_data(2, 1) = 6
% missing_phase_transitions contains the indices of rows in
% phase_transition_data indicating those phase jumps that did not occur in
% phase_transitions.
missing_phase_transitions = ...
    phase_transition_data(missing_phase_transitions_idx, 1);

if any(subcarrier_states(missing_phase_transitions - 1) ~= -1)
    warning(['Phase transitions detected from phase transition ', ...
        'correlation missing \n in phase transitions from single ', ...
        'subcarrier correlation.\n', ...
        'Phase transitions missing at subcarrier indices:\n\t%s\n'], ...
        sprintf('%d ', missing_phase_transitions))

    check_passed = false;
end


end


function state_determination_error = dual_channel_error_handling( ...
    return_value_re, return_value_im, ...
    initial_subcarrier_state_re, initial_subcarrier_state_im, ...
    subcarrier_indices_re, subcarrier_indices_im)

% function state_determination_error = dual_channel_error_handling( ...
%    return_value_re, return_value_im, ...
%    initial_subcarrier_state_re, initial_subcarrier_state_im, ...
%    subcarrier_indices_re, subcarrier_indices_im)
%
% DESCRIPTION: Function for printing channel dependent error messages when
%   using two channel mode.
%
%
% INPUT:
%  return_value_re..... Return value from real baseband channel
%  return_value_im..... Return value from imaginary baseband channel
%  initial_subcarrier_state_re.....Initial subcarrier state for real
```

```
%       baseband channel
%  initial_subcarrier_state_im..... Initial subcarrier state for imaginary
%       baseband channel
%  subcarrier_indices_re.....Subcarrier indices for real baseband channel
%  subcarrier_indices_im.....Subcarrier indices for imaginary baseband
%       channel
%
%
% OUTPUT:
%   state_determination_error.....Boolean variable indicating whether an
%       error occurred or not
%

state_determination_error = false;

if return_value_re == -1 || return_value_im == -1
    state_determination_error = true;
    return
end

if initial_subcarrier_state_re ~= initial_subcarrier_state_im

    warning(['Initial subcarrier states from real and imaginary ', ...
        'channel of the complex\nbaseband signal does not match.\n', ...
        'Initial subcarrier state of real channel: %d\n', ...
        'Initial subcarrier state of imaginary channel:%d\n'], ...
        initial_subcarrier_state_re, initial_subcarrier_state_im)

    state_determination_error = true;
    return
end

if length(subcarrier_indices_re) ~= length(subcarrier_indices_im)
    warning(['Real- and imaginary channel of the complex baseband ', ...
        'signal does not contain\nthe same number of subcarrier ', ...
        'periods.\n', ...
        'Real channel contains &d subcarrier periods.\n', ...
        'Imaginary channel contains %d subcarrier periods.'], ...
        length(subcarrier_indices_re) - 1, ...
        length(subcarrier_indices_im) - 1)
    state_determination_error = true;
    return
end

end
```

N.12.16 get_Vlma_limit.m

```
function [V_lma_min,V_lma_max, h_field] = get_Vlma_limit(V_calcoil, ...
    PICC_class)

% function [V_lma_min,V_lma_max, h_field] = get_Vlma_limit(V_calcoil, ...
%    PICC_class)
%
% DESCRIPTION: computes minimum PICC load modulation limits
%
% INPUT:
%       V_calcoil....rms value of the calibration coil voltage
%       PICC_class...class of the PICC {1, 2, 3, 4, 5, 6}
% OUTPUT:
%       V_lma_min...minimum required load modulation amplitude for the
%                   field strength and the PICC class in mV(p)
%       V_lma_max...maximum required load modulation amplitude for the
%                   field strength and the PICC class in mV(p)
%       h_field.....compute field strength form the unmodulated carrier
%                   in A/m(rms)

switch PICC_class

    case 2 % Use Test PCD Assembly 1
```

```
        h_field = V_calcoil / 0.318; % field strength [A/m]
        if h_field < 1.5
            V_lma_min = min([14, 22 / sqrt(1.5)]);
        elseif h_field > 8.5
            V_lma_min = min([14, 22 / sqrt(8.5)]);
        else
            V_lma_min = min([14, 22 / sqrt(h_field)]);
        end
        V_lma_max = 90;
    case 3 % Use Test PCD Assembly 1
        h_field = V_calcoil / 0.318; % field strength [A/m]
        if h_field < 1.5
            V_lma_min = min([14, 22 / sqrt(1.5)]);
        elseif h_field > 8.5
            V_lma_min = min([14, 22 / sqrt(8.5)]);
        else
            V_lma_min = min([14, 22 / sqrt(h_field)]);
        end
        V_lma_max = 80;
    case 4 % Use Test PCD Assembly 2
        h_field = V_calcoil / 0.118; % field strength [A/m]
        if h_field < 2.0
            V_lma_min = min([18, 40 / sqrt(2)]);
        elseif h_field > 12
            V_lma_min = min([18, 40 / sqrt(12)]);
        else
            V_lma_min = min([18, 40 / sqrt(h_field)]);
        end
        V_lma_max = 100;
    case 5 % Use Test PCD Assembly 2
        h_field = V_calcoil / 0.118; % field strength [A/m]
        if h_field < 2.5
            V_lma_min = min([14, 34 / sqrt(2.5)]);
        elseif h_field > 14
            V_lma_min = min([14, 34 / sqrt(14)]);
        else
            V_lma_min = min([14, 34 / sqrt(h_field)]);
        end
        V_lma_max = 90;
    case 6 % Use Test PCD Assembly 2
        h_field = V_calcoil / 0.118; % field strength [A/m]
        if h_field < 4.5
            V_lma_min = min([7, 26 / sqrt(4.5)]);
        elseif h_field > 18
            V_lma_min = min([7, 26 / sqrt(18)]);
        else
            V_lma_min = min([7, 26 / sqrt(h_field)]);
        end
        V_lma_max = 80;
    otherwise % Use Test PCD Assembly 1
        if PICC_class ~= 1
            fprintf(['No valid PICC class selected. The default PICC ',...
                'class 1 will be used!\r\n'])
        end
        h_field = V_calcoil / 0.318; % field strength [A/m]
        if h_field < 1.5
            V_lma_min = 22 / sqrt(1.5); % mV(p)
        elseif h_field > 7.5
            V_lma_min = 22 / sqrt(7.5);% mV(p)
        else
            V_lma_min = 22 / sqrt(h_field);% mV(p)
        end
        V_lma_max = 100;
end

end
```

N.12.17 make_periodic_frame.m

```
function [strt_idx, stp_idx] = make_periodic_frame(in, ...
    samples_per_carrier)

% function [strt_idx, stp_idx] = make_periodic_frame(in, ...
%   samples_per_carrier)
%
% DESCRIPTION: cuts parts at the end and beginning of signal in order to
%              have a periodic signal over the frame (leakage effect)
%
% INPUT:
%       in... HF signal
%       samples_per_carrier...Number of samples per 13.56 MHz carrier
%           period
%
% OUTPUT:
%       strt_idx...index of new starting point
%       stp_idx...index of new stop index

N = length(in);
obs_win = ceil(samples_per_carrier * 3);
idx_range_strt = 1 : obs_win;
idx_range_end = (N - obs_win + 1) : N;

% find maxima at the beginning
delt_delt_strt = diff(sign(diff(in(idx_range_strt))));
strt_max = find(delt_delt_strt(round(samples_per_carrier) : end) ...
    == -2, 1, 'first');
strt_max = strt_max + round(samples_per_carrier) - 1;

delt_delt_end = diff(sign(diff(in(idx_range_end))));
end_max = find(delt_delt_end == -2, 1, 'last');

strt_idx = strt_max - round(samples_per_carrier / 4) + 1;
stp_idx = end_max - round(samples_per_carrier / 4) + idx_range_end(1) - 1;
```

N.12.18 new_findpeaks.m

```
function [vals, locs] = new_findpeaks(sig)

% function [vals, locs] = new_findpeaks(sig)
%
% DESCRIPTION: Function for determining the local maxima in the passed
%              signal
%
% INPUT:
%       sig...The signal whos maxima should be determined
%
%
% OUTPUT:
%       vals...Values of the local maxima
%       locs...Indices of the local maxima

sig = sig(:);
dsd = diff(sign(diff(sig)));

locs =  find(dsd == -2) + 1;
vals = sig(locs);

end
```

N.12.19 peak_envelope.m

```
function [env_up, env_lo] = peak_envelope(signal, start_idx, end_idx)

% function [env_up, env_lo] = peak_envelope(signal, start_idx, end_idx)
%
% DESCRIPTION: Function for computing the upper and lower peak envelopes
```

```
%
%               The function calculates the local minima and maxima of the
%               mean free signal in the area specified by start_idx and
%               end_idx. The local extrema are interpolated using linear
%               interpolation.
%
% INPUT:
%       signal.............................Signal whose peak envelope for a
%                                          specific area should be
%                                          calculated
%       start_idx..........................Start of the area of the signal
%                                          whose peak envelope should be
%                                          calculated
%       end_idx............................End of area
% OUTPUT:
%
%       env_up.....Upper envelope
%       env_lo.....Upper envelope

mean_sig = mean(signal);
mean_free_signal = signal - mean_sig;

% Find, cleanup peaks and interpolate the extrema using linear
% interpolation
sig = mean_free_signal;
[vals_up, idx_up] = new_findpeaks(sig);
idx_up(vals_up < 0.1 * max(signal)) = [];
vals_up(vals_up < 0.1 * max(signal)) = [];

env_up = interp1(idx_up, vals_up, 1 : length(signal));


sig = mean_free_signal;
[vals_lo, idx_lo] = new_findpeaks(-sig);
idx_lo(vals_lo < 0.1 * max(-signal)) = [];
vals_lo(vals_lo < 0.1 * max(-signal)) = [];
vals_lo = -vals_lo;

env_lo = interp1(idx_lo, vals_lo, 1 : length(signal));


%Cut out specified areas
env_up = env_up(start_idx : end_idx);
env_lo = env_lo(start_idx : end_idx);

env_up = env_up(:);
env_lo = env_lo(:);

%Remove nan's from interpolation
env_up(isnan(env_up)) = 0;
env_lo(isnan(env_lo)) = 0;

%Align envelopes to original signal
env_up = env_up(:) + mean_sig;
env_lo = env_lo(:) + mean_sig;

end
```

N.12.20 phase_drift_within_and_over_state.m

```
function [max_intraState_PD, max_interState_PD, h_fig] = ...
    phase_drift_within_and_over_state(MS1_sig_m, MS2_sig_m)
% function [max_intraState_PD, max_interState_PD] = ...
% phase_drift_within_and_over_state(MS1_sig_m, MS2_sig_m)
%
% DESCRIPTION: computes the phase drift within each state occurrence
% (intrastate phase drift) and the phase drift over the complete PICC
% response(interstate phase drift)
%
% INPUT:
```

```
%       MS1_sig_m...row: state occurrences MS1, col: samples of each
%                  state occurrence
%       MS2_sig_m...row: state occurrences MS2, col: samples of each
%                  state occurrence
% OUTPUT:
%
% max_intraState_PD...the phase drift within each state occurrence
% max_interState_PD...the phase drift over all state occurrences

PLOT = 1;

% MS1_sig_m matrix: column: samples of each state; row: states
% [state 1: Samp 1, Samp 2, Samp 3, Samp 4, ...
%  state 2: Samp 1, Samp 2, Samp 3, Samp 4, ...
%  state 3: Samp 1, Samp 2, Samp 3, Samp 4, ...
%  state 4: Samp 1, Samp 2, Samp 3, Samp 4, ...
%  ...
%  state N: Samp 1, Samp 2, Samp 3, Samp 4, ... ]


% adjust number of MS1 and MS2 states to be equal
N_ms1 = size(MS1_sig_m, 1);
N_ms2 = size(MS2_sig_m, 1);

% compute complex difference (MS1-MS2)
if N_ms1 > N_ms2
   delta = MS1_sig_m(1 : N_ms2, :) - MS2_sig_m;
   Nphi_delta = N_ms2;
else
   delta = MS1_sig_m - MS2_sig_m(1 : N_ms1, :);
   Nphi_delta = N_ms1;
end

% Compute the angle of MS1-MS2
phi_delta = unwrap(unwrap(angle(delta), [], 1), [], 2) * 180 / pi;

% ------------------------------------------------------------------------ %
% Intrastate PD
% find max delta phase and min delta phase in each state and subtract
% This corresponds to the phase change within each MS1-MS2 occurrence
[max_vec, max_idx] = max(phi_delta, [], 2);
[min_vec, min_idx] = min(phi_delta, [], 2);
slope = sign(max_idx - min_idx);


state_PD = (max_vec - min_vec).*slope;
% the intrastate PD is the difference between min and max of all
% occurrences, since a drift can have a positive and negative slope
min_state_PD = min(state_PD);
max_state_PD = max(state_PD);
if min_state_PD >= 0 && max_state_PD >= 0
    max_intraState_PD = abs(max_state_PD);
elseif min_state_PD < 0 && max_state_PD >= 0
    max_intraState_PD = abs(max(state_PD) - min(state_PD));
elseif min_state_PD < 0 && max_state_PD < 0
    max_intraState_PD = abs(min_state_PD);
end


% ------------------------------------------------------------------------ %
% Interstate PD
% ------------ %
avg_delta =  mean(delta, 2); % avg complex vector difference of each state
avg_phi = unwrap(unwrap(angle(avg_delta), [], 1), [], 2) * 180 / pi;
max_interState_PD = abs(max(avg_phi) - min(avg_phi));

if PLOT
    lw = 2; % plot: line width
    h_fig = figure('color', [1 1 1]);
    scrsz = get(0, 'ScreenSize');
    set(h_fig, 'position',...
```

```
        [scrsz(3) / 2, scrsz(4) * .47, scrsz(4) * .6, scrsz(4) * .4])
    MS = 7; % marker size
    LW = 2; % line width

    figure(h_fig),
    subplot(211);
    plot(state_PD, 'k', 'linewidth', lw)
    title('\Delta \Phi_{LM}')
    grid on
    xlabel('Subcarrier periods')
    ylabel(['Argument (', char(176), ')'])
    axis tight
    %%%%%%%%%%%%%%%%%%%%
    subplot(212);
    plot(avg_phi, 'k', 'linewidth', lw)
    title('\Phi_{LM, MEAN}')
    grid on
    xlabel('Subcarrier periods')
    ylabel(['Argument (', char(176), ')'])
    axis tight
end
```

N.12.21 plot_time_continous_const_plot.m

```
function h_fig = plot_time_continous_const_plot(U_sig, trans, MS1_sig, ...
    MS2_sig)

% function h_fig = plot_time_continous_const_plot(U_sig, trans, MS1_sig,
%   MS2_sig)
%
% DESCRIPTION: Plot function for Constellation diagram visualizing the
%              states MS1, MS2, US overlayed over the complex time
%              continuous signal
%
% INPUT:
%       U_sig.....unmodulated carrier signal component in complex baseband
%       trans.....vector containing signal components of the transitions
%                 from MS1 to MS2 and vice versa
%       MS1_sig.....Samples of signal belonging to state MS1 occurrences
%       MS2_sig.....Samples of signal belonging to state MS2 occurrences
% OUTPUT:
%       h_fig.....figure handle


% plot PICC response in constellation diagram
h_fig = figure('color', [1 1 1]);
scrsz = get(0, 'ScreenSize');
set(h_fig, 'position',...
    [scrsz(3) / 16, scrsz(4) * .47, scrsz(4) * .6, scrsz(4) * .4])

MS = 7; % marker size
LW = 2; % line width

% plot time continuous signal
hold on; h_ax(4) = plot(1000 * real(trans), 1000 * imag(trans), '--');
set(h_ax(4), 'color', [0.5, 0.4, 0.5])

%plot state MS1 data
hold on; h_ax(2) = plot(1000 * real(MS1_sig), ...
                        1000 * imag(MS1_sig), 'x',...
                               'LineWidth', LW,...
                               'MarkerEdgeColor', ...
                                   [0.3,1,0],...
                               'MarkerFaceColor', ...
                                   [0.3,1,0],...
                               'MarkerSize', MS);
%plot state MS2 data
hold on; h_ax(3) = plot(1000 * real(MS2_sig), ...
                        1000 * imag(MS2_sig), 'x',...
                               'LineWidth', LW,...
```

```
                                'MarkerEdgeColor', 'b',...
                                'MarkerFaceColor', 'b',...
                                'MarkerSize', MS);


% plot first state MS1 of PICC response
hold on;
h_ax(5) = plot( 1000 * real(mean(MS1_sig(1 : 10))), ...
                1000 * imag(mean(MS1_sig(1:10))),...
                                        'x',...
                                        'LineWidth', LW,...
                                        'MarkerEdgeColor', 'r',...
                                        'MarkerFaceColor', 'r',...
                                        'MarkerSize', MS + 1);

% plot state US
h_ax(1) = plot(1000 * real(U_sig), 1000 * imag(U_sig), 'o',...
                                'LineWidth', LW,...
                                'MarkerEdgeColor', [0.83, 0.83, 0.83],...
                                'MarkerFaceColor', [0.83, 0.83, 0.83],...
                                'MarkerSize', MS);


title('Constellation diagram')
xlabel('In-phase (mV)')
ylabel('Q-phase (mV)')
grid on

legend(h_ax, 'State US', 'State MS1','State MS2', 'Transition',...
    'First state', 'location', 'EastOutside')
axis square
% place text within plot
x_dim = xlim;
d_x_dim = abs(diff(x_dim));
y_dim = ylim;
d_y_dim = abs(diff(y_dim));
% Scale axes to be equal
if d_x_dim > d_y_dim
    XMIN = x_dim(1);
    XMAX = x_dim(2);
    YMIN = (y_dim(1) + y_dim(2))/2 - d_x_dim/2;
    YMAX = (y_dim(1) + y_dim(2))/2 + d_x_dim/2;
else
    YMIN = y_dim(1);
    YMAX = y_dim(2);
    XMIN = (x_dim(1) + x_dim(2))/2 - d_y_dim/2;
    XMAX = (x_dim(1) + x_dim(2))/2 + d_y_dim/2;
end

xlim([XMIN XMAX])
ylim([YMIN YMAX])
set(gca, 'box', 'on')
```

N.12.22 Type_A_not_106_decoding.m

```
function [bit_states, ret_val] = ...
    Type_A_not_106_decoding(subcarrier_states, encoding, ...
    ignore_last_subcarrier)

% function [bit_states, ret_val] = ...
%    Type_A_not_106_decoding(subcarrier_states, encoding, ...
%    ignore_last_subcarrier)
%
% DESCRIPTION: Function for decoding of a Type A signal with a bitrate of
%              higher than 106 kBps
%
%
% INPUT:
%       subcarrier_states.....Vector containing the subcarrier states
%       encoding.... The encoding of the signal
```

```
%       ignore_last_subcarrier...FLAG, if 1b the last subcarrier will be
%           ignored
% OUTPUT:
%       bit_states.....Vector containing the bit states
%       ret_val.....If not 0, an error or warning was triggered
%                   during signal decoding

ret_val = 0;

encoding_parameters = get_encoding_parameters(encoding);
subcarrier_cycles_per_etu = encoding_parameters.subcarrier_cycles_per_etu;


% ----- Determination of the phase transitions
tmp1 = [subcarrier_states; subcarrier_states(end)];
tmp2 = [subcarrier_states(1) ; subcarrier_states];
phase_transitions = xor(tmp1, tmp2);
phase_transitions = phase_transitions(1 : end - 1);

% Vector from above contains one entry for each subcarrier where
% subcarriers with phase transitions are denoted by a '1', all others by a
% '0'.
%
% Now create a vector that contains the indices of subcarriers with phase
% transitions.
phase_transitions = find(phase_transitions == 1);


%-------------------------- Check SOC -----------------------------------
SOC_high_start = 1;
SOC_high_end = phase_transitions(1) - 1;

SOC_high_length = SOC_high_end - SOC_high_start + 1;

bit_sequence_start = phase_transitions(1) + subcarrier_cycles_per_etu;
one_state = subcarrier_states(1);
zero_state = ~one_state;

if SOC_high_length ~= 32
    warning(['Data encoding error. SOC high length must be 32 ', ...
        'subcarriers but measured length was %d subcarriers.'], ...
        SOC_high_length)
end

%------------------------ Bit decoding ---------------------------------
bit_sequence_length = length(subcarrier_states) - bit_sequence_start + 1;
number_of_bits = floor(bit_sequence_length / subcarrier_cycles_per_etu);

% If an none integer number of bits was detected decide whether this
% should be treated as an data encoding error or not.
%
% If the frame contains an integer number of etus without the last
% subcarrier and ignoring this subcarrier is allowed inform the user and
% continue decoding. Otherwise throw an error.
if mod(bit_sequence_length, subcarrier_cycles_per_etu)
    if ignore_last_subcarrier && ~mod(bit_sequence_length - 1, ...
            subcarrier_cycles_per_etu)


        fprintf(['Bit sequence contains %d subcarriers which is not ', ...
            'equal to an integer number of etus. (%.2f etus). \nAs ', ...
            'excluding the last subcarrier of the response is ', ...
            'allowed the script will decode %d bits ', ...
            '(%d subcarriers)'], ...
            bit_sequence_length, bit_sequence_length / ...
            subcarrier_cycles_per_etu, ...
            number_of_bits, number_of_bits * subcarrier_cycles_per_etu)

        % Calculate the new number of bits to decode.
        bit_sequence_length = bit_sequence_length - 1;
```

```
    else
        warning(['Bit sequence contains %d subcarriers which is not ', ...
            'equal to an integer number of etus. (%.2f etus). ', ...
            '\nProgram will decode %d bits. (%d subcarriers)'], ...
            bit_sequence_length, bit_sequence_length / ...
            subcarrier_cycles_per_etu, ...
            number_of_bits, number_of_bits * subcarrier_cycles_per_etu)

        % Calculate the new number of bits to decode.
        bit_sequence_length = floor(bit_sequence_length / ...
            subcarrier_cycles_per_etu) ...
            * subcarrier_cycles_per_etu;
    end
end

% Extract subcarrier containing the encoded data from the complete
% response
bit_sequence = subcarrier_states(bit_sequence_start : ...
    bit_sequence_start + bit_sequence_length - 1);

% Reshape the vector to a matrix with subcarrier_cycles_per_etu columns
% and one row for each bit.
subcarrier_states_in_bits = reshape(bit_sequence, ...
    subcarrier_cycles_per_etu, number_of_bits);
subcarrier_states_in_bits = subcarrier_states_in_bits';

% ----- Bit state determination
bit_states = -ones(number_of_bits, 1);

bit_states(all(subcarrier_states_in_bits == zero_state * ...
    ones(number_of_bits, subcarrier_cycles_per_etu), 2)) = zero_state;

bit_states(all(subcarrier_states_in_bits == one_state * ...
    ones(number_of_bits, subcarrier_cycles_per_etu), 2)) = one_state;

if any(bit_states == -1)
    warning(['The states of the following bits could not be ', ...
        'determined: ' , ...
        sprintf('%d ', find(bit_states == -1))])
    ret_val = -1;
end

%Bit boundary calculation
bit_boundaries = (0 : number_of_bits + 1) * ...
    subcarrier_cycles_per_etu + bit_sequence_start;
bit_boundaries = [bit_boundaries(1 : end - 1)', ...
    bit_boundaries(2 : end)' - 1];
end
```

N.12.23 Type_B_decoding.m

```
function [bit_states_in_characters, SOF_start, ret_val] = ...
    Type_B_decoding(subcarrier_states, encoding)

% function [bit_states_in_characters, SOF_start, ret_val] = ...
%    Type_B_decoding(subcarrier_states, encoding)
%
% DESCRIPTION: Function for decoding a Type B PICC response.
%              The function uses the subcarrier states to for decoding
%
%
% INPUT:
%       subcarrier_states.....Vector containing the subcarrier states
%       encoding..............The encoding of the signal
%
% OUTPUT:
%       bit_states_in_characters.....Matrix containing the subarrier
%                                    states of each character as rows
%       SOF_start.....Number of subcarrier where the SOF sequence starts
%       ret_val.....Error flag, If not zero, an error occurred
```

```
ret_val = 0;
% For Type B signals with a bitrate of 106kBps SOF and/or EOF can be
% suppressed.
if strcmp(encoding.bitrate, '106')
    EOF_SOF_suppression_possible = 1;
else
    EOF_SOF_suppression_possible = 0;
end

encoding_parameters = get_encoding_parameters(encoding);
subcarrier_cycles_per_etu = encoding_parameters.subcarrier_cycles_per_etu;

% Determine the subcarrier state corresponding to the bitstates of logic 0
% and logic 1.
%
% According to ISO/IEC 14443-3 the subcarrier state corresponding to logic
% zero is the state of the first subcarrier in the response.
% The subcarrier state corresponding to a bitstate of logic 1 is the
% inverse state of the first subcarrier
one_state = subcarrier_states(1);
zero_state = ~one_state;


% Get subcarriers with subcarrier phase transitions
tmp1 = [subcarrier_states; subcarrier_states(end)];
tmp2 = [subcarrier_states(1) ; subcarrier_states];
phase_transitions = xor(tmp1, tmp2);
phase_transitions = phase_transitions(1 : end - 1);

phase_transitions = [find(phase_transitions == 1), ...
    phase_transitions(phase_transitions == 1)];

% Determine the type of the phase transition.
for k = 1 : length(phase_transitions)
    state_k = subcarrier_states(phase_transitions(k, 1));
    state_k_m1 = subcarrier_states(phase_transitions(k, 1) - 1);

    %Rising edge
    if state_k_m1 == zero_state && state_k == one_state
        phase_transitions(k, 2) = 2;

    %Falling edge
    else
        phase_transitions(k, 2) = 1;
    end
end

% Check the lengths of TR1 and SOF and determine the start of the bit
% sequence
[bit_sequence_start, ~, SOF_start, rv] = check_TR1_SOF(...
    phase_transitions, encoding);
if rv ~= 0
    warning(['An error occurred while applying TR1 and SOF check. \n', ...
        'Terminating script.'])
    ret_val = -1;
end

[bit_states_in_characters, character_boundaries, EOF_start, ~, rv] = ...
    bit_decoding_and_EOF_check(...
    subcarrier_states, phase_transitions, bit_sequence_start, encoding,...
    zero_state, EOF_SOF_suppression_possible);
if rv ~= 0
    warning(['An error occurred while response decoding and EGT ', ...
        'check. \nTerminating script.'])
    ret_val = -1;
    return
end

%-------------------------- EGT length check ----------------------------
[number_of_characters, ~] = size(bit_states_in_characters);
```

```
% One row for each EGT in the whole frame. First column: EGT start,
% Second column: EGT end, third column: EGT length
EGT_data = -ones(number_of_characters, 3);


% Calculation of EGT between characters
for k = 1 : number_of_characters - 1
    EGT_start = character_boundaries(k, 2) + 1;
    EGT_end = character_boundaries(k + 1, 1);
    EGT_length = EGT_end - EGT_start;

    if EGT_length > 2 * subcarrier_cycles_per_etu
        warning(['Measured EGT between character %d and %d is ', ...
            '%4.2f etus (%d subcarrier periods) \nand therefore ', ...
            'longer than the maximum allowed length of 2 etus ', ...
            '(%d subcarrier periods).'], ...
            k, k + 1, EGT_length / subcarrier_cycles_per_etu, ...
            EGT_length, 2 * subcarrier_cycles_per_etu)
        ret_val = -1;
    end

    EGT_data(k, :) = [EGT_start, EGT_end, EGT_length];
end

%EGT between last bit and EOF
EGT_start = character_boundaries(end, 2) + 1;
EGT_end = EOF_start;
EGT_length = EGT_end - EGT_start;

if EGT_length > 2 * subcarrier_cycles_per_etu
    warning(['EGT between last character and EOF is longer than ', ...
        '2 etus (%d subcarrier periods). Measured EGT is %4.2f etus ', ...
        '(%d subcarrier periods).'], ...
        2 * subcarrier_cycles_per_etu, EGT_length / ...
        subcarrier_cycles_per_etu, EGT_length)
    ret_val = -1;
end

EGT_data(end, :) = [EGT_start, EGT_end, EGT_length];
end

function [bit_sequence_start, SOF_suppressed, SOF_Start, ret_val] = ...
    check_TR1_SOF(phase_transitions, encoding)

% function [bit_sequence_start, SOF_suppressed, SOF_Start, ret_val] = ...
%    check_TR1_SOF(phase_transitions, encoding)
%
% DESCRIPTION: Function for checking the lengths of TR1, SOF high and SOF
%              low.
%
%              ISO/IEC 14443-3 defines the absolute minimum value of
%              TR1 as 16 subcarriers for a bitrate of 106 kBps and 8
%              subcarriers for all other bitrates.
%
%              SOF low must be between 10 and 11 etus and SOF high between
%              2 and 3 etus long. In case of a signal with a bitrate of
%              106 kBps SOF and/or EOF can be suppressed.
%              The algorithm assumes SOF to be suppressed when the
%              distance between the first and the second phase transition,
%              which is supposed to be the length of SOF low, is shorter
%              than 9 etus. (This would correspond to a character
%              containing 0x00).
%
%
% INPUT:
%       phase_transitions.....Nx2 matrix containing in the first column
%                             the index of the phase transition and in
%                             the second column the type of the
%                             phase transition (2 : rising, 1: falling)
%       encoding.....The encoding of the signal
```

```
%
% OUTPUT:
%       bit_sequence_start.....Index of the start of the bit sequence
%       SOF_suppressed.....Flag (0 or 1) indicating that SOF is suppressed
%       SOF_Start.....Index of the start of the SOF sequence
%       ret_val.....If not zero, an error or warning was triggered

ret_val = 0;

encoding_parameters = get_encoding_parameters(encoding);
subcarrier_cycles_per_etu = encoding_parameters.subcarrier_cycles_per_etu;

% For Type B signals with a bitrate of 106 kBps SOF and/or EOF can be
% suppressed.
%
% According to ISO/IEC 14443-3:2018 7.10.3.2 the length of TR1 may be
% reduced to 16 subcarrier cycles for a bitrate of 106 kBps or 8
% subcarrier cycles for all other bitrates
if strcmp(encoding.bitrate, '106')
    EOF_SOF_suppression_possible = 1;
    TR1_min = 16;
else
    TR1_min = 8;
    EOF_SOF_suppression_possible = 0;
end

%------------- Check if TR1 is longer than its minimum value -------------
TR1_length = phase_transitions(1, 1);


if TR1_length < TR1_min
    warning(['Detected length of TR1 is shorter than its minimum ', ...
        'required length of %d subcarrier periods.\nThe actual ', ...
        'length (distance to first phase transition) is %d ', ...
        'subcarrier periods.'], ...
        TR1_min, TR1_length)
    ret_val = -1;
else
    fprintf(['Detected length of TR1: %d subcarrier ', ...
        'periods\n\r'], TR1_length)
end
%----- Check if SOF high and SOF low are within their specified ranges----

% According to ISO/IEC 14443:2018 SOF and/or EOF can be suppressed for
% signals with a bitrate of 106 kBps.
%
% The algorithm checks if the distance between the first and the second
% phase transition (= SOF low length) and between the second and third
% phase transition (= SOF high length) are within the limits as defined in
% ISO/IEC 14443
%
% If the SOF low length is shorter than 10 etus for a signal with a
% bitrate of 106 kBps, the algorithm assumes that SOF is suppressed.


%----- SOF low length check
SOF_low_length = phase_transitions(2, 1) - phase_transitions(1, 1);
SOF_suppressed = 0;

if SOF_low_length < 10 * subcarrier_cycles_per_etu

    %Suppression possible only for 106kBps responses
    if EOF_SOF_suppression_possible && SOF_low_length < 9 * ...
            subcarrier_cycles_per_etu
        SOF_suppressed = 1;
        fprintf(['Length of SOF low is %4.2f etus (%d subcarrier ', ...
            'periods). \nBecause the response has a bitrate of 106 ', ...
            'kBps and the length \nof SOF low is shorter than 9 etus ',...
            '(72 subcarrier periods) SOF is assumed \nto be ', ...
            'suppressed.'], ...
            SOF_low_length, double(SOF_low_length) / ...
```

```
            subcarrier_cycles_per_etu)
    else
        warning(['Length of SOF low is %4.2f etus (%d subcarrier ', ...
            'periods) which is shorter \nthan its minimum required ', ...
            'length of 10 etus (%d subcarrier periods)'], ...
            double(SOF_low_length) / subcarrier_cycles_per_etu, ...
            SOF_low_length, 10 * subcarrier_cycles_per_etu)
        ret_val = -1;
    end


elseif SOF_low_length > 11 * subcarrier_cycles_per_etu
    warning(['Length of SOF low is %4.2f etus (%d subcarrier ', ...
        'periods) which is longer \nthan its maximium allowed ', ...
        'length of 11 etus (%d subcarrier periods)'], ...
        double(SOF_low_length) / ...
        subcarrier_cycles_per_etu, SOF_low_length, ...
        11 * subcarrier_cycles_per_etu)
        ret_val = -1;
end


SOF_Start = phase_transitions(1,1);


%----- SOF high length check
SOF_high_length = phase_transitions(3, 1) - phase_transitions(2, 1);

if ~SOF_suppressed
    if SOF_high_length < 2 * subcarrier_cycles_per_etu
        warning(['Length of SOF high is %4.2f etus (%d subcarrier ', ...
            'periods) which is shorter \nthan its minimum required ', ...
            'length of 2 etus (%d subcarrier periods)'], ...
            double(SOF_low_length) / ...
            subcarrier_cycles_per_etu, SOF_low_length, ...
            2 * subcarrier_cycles_per_etu)
        ret_val = -1;

    elseif SOF_high_length > 3 * subcarrier_cycles_per_etu
        warning(['Length of SOF high is %4.2f etus (%d subcarrier ', ...
            'periods) which is longer \nthan its maximum allowed ', ...
            'length of 3 etus (%d subcarrier periods)'], ...
            double(SOF_low_length) / ...
            subcarrier_cycles_per_etu, SOF_low_length, ...
            3 * subcarrier_cycles_per_etu)
        ret_val = -1;
    end

end

%----- Check if phase transition edge at the first start bit is a
% 'falling' edge.
% A falling edge is denoted by the value '1' in the second column of
% phase_transitions, a rising edge by the value '2'.

% In case of a SOF suppressed response the first phase transition denotes
% the transition from TR1 to the first start bit.
if SOF_suppressed
    if phase_transitions(1,2) ~= 1

        % If no falling edge was detected, search for the next one and
        % start decoding procedure at this index.
        tmp = find(phase_transitions(2 : end, 2) == 1, 1, 'first');
        bit_sequence_start = phase_transitions(tmp + 1, 1);

        warning(['First phase transition in SOF suppressed signal ', ...
            'must be a \nfalling edge (first start bit) but a rising ',...
            'edge was detected.\nTherefore the decoding procedure ', ...
            'will start at the next falling \nedge located at ', ...
            'subcarrier %d.'], bit_sequence_start)
        ret_val = -1;
```

```
    else
        bit_sequence_start = phase_transitions(1,1);
    end


% In case of a not SOF suppressed response the third phase transition
% denotes the transition from SOF high to the first start bit.
else
    if phase_transitions(3,2) ~= 1
        % If no falling edge was detected, search for the next one and
        % start decoding procedure at this index.
        tmp = find(phase_transitions(4 : end, 2) == 1, 1, 'first');
        bit_sequence_start = phase_transitions(tmp + 3, 1);

        warning(['First phase transition in a SOF suppressed ', ...
            'response must be a \nfalling edge (first start bit) but ',...
            'a rising edge was detected.\n',...
            'Therefore the decoding procedure will start at the ', ...
            'next falling \nedge located at subcarrier %d.'], ...
            bit_sequence_start)
        ret_val = -1;

    else
        bit_sequence_start = phase_transitions(3,1);
    end
end
end

function [bit_states_in_characters, character_boundaries, EOF_start, ...
    EOF_end, ret_val] = ...
    bit_decoding_and_EOF_check(subcarrier_states, phase_transitions, ...
    bit_sequence_start, encoding, zero_state, ...
    EOF_SOF_suppression_possible)

% function [bit_states_in_characters, character_boundaries, EOF_start, ...
%    EOF_end, ret_val] = ...
%    bit_decoding_and_EOF_check(subcarrier_states, phase_transitions, ...
%    bit_sequence_start, encoding, zero_state, ...
%    EOF_SOF_suppression_possible)
%
% DESCRIPTION: Function for determining the encoding of each bit in each
%              character. Additionally the EOF sequence is checked after
%              the last character.
%
%              The last character is normally invalid because it contains
%              10 etus of logic 0 which corresponds to the EOF low
%              sequence.
%              A valid last character is only allowed of EOF suppression
%              is possible.
%
%              The algorithm checks also if only one phase transition
%              after EOF low and if it is a rising edge.
%              (See ISO/IEC 14443-3)
%
%
%
% INPUT:
%       subcarrier_states.....Vector containing the subcarrier states
%       phase_transitions.....Nx2 matrix containing the indices and types
%                             of the phase transitions
%       bit_sequence_start.....The start index of the bit sequence in the
%                              response
%       encoding.....The encoding of the signal
%       zero_state.....Subcarrier state corresponding to logic 0
%       EOF_SOF_suppression_possible.....Flag (0 or 1) indicating if
%                                        suppression of SOF and/or EOF is
%                                        possible
%
% OUTPUT:
%       bit_states_in_characters.....Matrix with <number_of_characters>
%                                    rows and 10 columns where each row
```

```
%                                   contains the state of the
%                                   corresponding bit (including Start-
%                                   and Stop-bit)
%       character_boundaries.....Nx2 matrix containing the character
%                                boundaries
%       EOF_start.....Start index of the EOF sequence
%       EOF_end.....End index of the EOF sequence
%       ret_val.....If not zero, an error or warning was triggered


% Pre- initialization of return values
EOF_start = -1;
EOF_end = -1;
bit_states_in_characters = -1;
character_boundaries = -1;

ret_val = 0;

encoding_parameters = get_encoding_parameters(encoding);
subcarrier_cycles_per_etu = encoding_parameters.subcarrier_cycles_per_etu;


% ------------------ Data decoding

% Maximum possible number of bits (e.g. if no EGT occurs)
max_number_of_bits = ceil(length(subcarrier_states) / ...
    (10 * subcarrier_cycles_per_etu));

subcarrier_states_in_character = -ones(max_number_of_bits, ...
    10 * subcarrier_cycles_per_etu);

character_boundaries = -ones(max_number_of_bits, 2);



character_start = bit_sequence_start;
character_end = character_start + 10 * subcarrier_cycles_per_etu - 1;
for k = 1 : max_number_of_bits

    subcarrier_states_in_character(k, :) = subcarrier_states(...
        character_start : character_end);
    character_boundaries(k, :) = [character_start, character_end];



    [character_start, rv] = find_next_start_bit(phase_transitions, ...
        character_end);
    if rv ~= 0
        ret_val = -1;
        return
    end
    %End of frame reached
    if character_start == -1
        break;
    end
    character_end = character_start + 10 * subcarrier_cycles_per_etu - 1;

    if character_end > length(subcarrier_states)
        warning(['Found start bit at subcarrier %d which results ', ...
            'in an end of character at %d subcarrier periods but the ',...
            'response contains only %d subcarrier periods.'], ...
            character_start, character_end, length(subcarrier_states));
        ret_val = -1;
        break;
    end
end

% Remove entries without information (rows containing only -1)
% from subcarrier_states_in_character.
subcarrier_states_in_character(subcarrier_states_in_character(:, 1) ...
    == -1, :) = [];
```

```
character_boundaries(character_boundaries(:, 1) == -1, :) = [];

% Determination of bit states from subcarrier states in each character
bit_states_in_characters = get_bit_states_in_character(...
    subcarrier_states_in_character, ...
    subcarrier_cycles_per_etu, zero_state);


% ------------------ EOF check
% When EOF is not suppressed, the last detected character is detected as
% invalid because of the at least 10 etus of logic low in EOF.
%
% EOF is determined as valid if all subcarrier states in this last
% "invalid" character have the state corresponsing to logic zero, only one
% rising edge occurs after the end of the EOF low sequence and the length
% of the EOF low sequence is >= 10 etus and <= 11 etus
phase_transitions_after_last_character = phase_transitions( ...
    phase_transitions(:, 1) > character_boundaries(end, 2), :);

if bit_states_in_characters(end, 11) == -1

    % [ 2,0,0,0,0,0,0,0,0,-1] denotes 10 etus of logic 0 in one character.
    % The entry
    % of the last column is ignored as it denotes if the complete
    % character is valid or not.
    %
    % Also check if all subcarriers in the last character, which is
    % supposed to be the EOF sequence, are equal to the logic zero
    % subcarrier state.
    if isequal(bit_states_in_characters(end, 1 : 10), ...
            [2,0,0,0,0,0,0,0,0,-1]) && ...
            all(subcarrier_states(character_boundaries(end, 1) : ...
            character_boundaries(end, 2)) == zero_state)

        % EOF sequence detected. Check if sequence is valid
        EOF_start = character_boundaries(end, 1);
        EOF_end = length(subcarrier_states);

        % Remove the entries for the EOF sequence from the matrix of bit
        % states and from the matrix of character boundaries
        bit_states_in_characters(end, :) = [];
        character_boundaries(end, :) = [];

        % After EOF low, one phase transition is supposed to occur
        if isequal(size(phase_transitions_after_last_character), [1,2])
            [EOF_length_ok] = check_EOF_length(EOF_start, ...
                phase_transitions_after_last_character, ...
                subcarrier_cycles_per_etu);
            if ~EOF_length_ok
                ret_val = -1;
            end

        % No phase transitions after EOF low
        elseif isempty(phase_transitions_after_last_character)
            warning(['EOF sequence not valid because after EOF low ', ...
                'no phase transition occurs.'])

        % More than one phase transition after EOF low
        else
            warning(['EOF sequence not valid because more than one ', ...
                'phase transition occur after EOF low.'])
            ret_val = -1;
        end
    else
        warning(['Last detected character is supposed to be EOF but ', ...
            'has bit states different from logic 0 in its 10 etus.'])
        ret_val = -1;
    end

else
    if ~EOF_SOF_suppression_possible
```

```
        warning(['Regular character was detected at end of frame ', ...
            'instead of EOF sequence although EOF suppression is ', ...
            'not possible.'])
        ret_val = -1;
    end
end
end

function [next_character_start, ret_val] = find_next_start_bit( ...
    phase_transitions, current_character_end)

% function [next_character_start, ret_val] = find_next_start_bit( ...
%    phase_transitions, current_character_end)
%
% DESCRIPTION: Function for finding the next start bit after the end of a
%              character
%
%              The function checks if the next phase transition after the
%              end of the current character denoted a falling edge. If not
%              a warning is printed and the next falling edge is returned
%              as start of the next character.
%
%
%
% INPUT:
%       phase_transitions.....Nx2 matrix containing the indices and types
%                             of the phase transitions
%       current_character_end.....Index of the end of the current
%                                 character
%
% OUTPUT:
%       next_character_start.....Index of the next start bit
%       ret_val.....If not zero, an error or warning was
%                   triggered

ret_val = 0;
% Phase transition for searching for the next start bit are all phase
% transitions after the the of the current stop bit.
phase_transitions_for_search = phase_transitions( ...
    phase_transitions(:, 1) > current_character_end, :);


% No phase transition, or only one remaining rising edge found -> No
% character found. This case denotes the EOF sequence
if isempty(phase_transitions_for_search) || ...
        (length(phase_transitions_for_search(:, 1)) == 1 && ...
        phase_transitions_for_search(1,2) == 2)
    next_character_start = -1;

% The next phase transition must be a falling one. Otherwise a data
% encoding occurred.
elseif phase_transitions_for_search(1,2) ~= 1
    tmp = find(phase_transitions_for_search(:, 2) == 1, 1, 'first');
    next_character_start = phase_transitions_for_search(tmp, 1);

    warning(['Error: Phase transition located at subcarrier %d is ', ...
        'expected to be a falling edge because it is \nsupposed to ', ...
        'be the begin of the start bit of a new character, but it ', ...
        'is a rising edge.'], ...
        phase_transitions_for_search(1,1))
    ret_val = -1;
else
    next_character_start = phase_transitions_for_search(1,1);
end
end

function bit_states_in_character = get_bit_states_in_character(...
    subcarrier_states_in_character, ...
    subcarrier_cycles_per_etu, zero_state)

% function bit_states_in_character = get_bit_states_in_character(...
```

```
%    subcarrier_states_in_character, ...
%    subcarrier_cycles_per_etu, zero_state)
%
% DESCRIPTION: Function for extracting the single bits of the subcarrier
%              sequence in each character passing it to a function for bit
%              state determination and storing it to a matrix containing
%              the bit states of each character as rows
%
%
%
% INPUT:
%       subcarrier_states_in_character.....Matrix containing the
%                                          subcarrier states in each
%                                          character as rows
%       subcarrier_cycles_per_etu.....Number of subcarrier cycles in one
%                                     etu
%       zero_state.....Subcarrier state corresponding to logic 0
%
% OUTPUT:
%       bit_states_in_character.....N x 11 Matrix containing the bit
%                                   states of each character as rows
%                                   including Start- and Stop-bit.
%                                   The 11th column contains the state of
%                                   the whole character (0 if no error in
%                                   character, -1 if the character
%                                   contains an invalid bit)
%

[number_of_characters, ~] = size(subcarrier_states_in_character);

% One row for each character. One column for each bit containing the state
% of the bit.
%
% Bit states:
% 0 : Logic 0 (Data bit)
% 1 : Logic 1 (Data bit)
% 2 : Start bit
% 3 : Stop bit
% -1: Invalid bit (e.g. not all subcarriers in bit have the same state or
%     start bit is logic 1
%
% The last 11th column contains the state of the whole character.
% 0 : Character OK
% -1: Character not OK (Wrong logic states of Start/Stop bit, etc.)
%
bit_states_in_character = -ones(number_of_characters, 11);



for character_idx = 1 : number_of_characters

    for bit_idx = 1 : 10 % 8 Bits + Start and Stop bit

        subcarrier_states_start = (bit_idx - 1) * ...
            subcarrier_cycles_per_etu + 1;
        subcarrier_states_end = subcarrier_states_start + ...
            subcarrier_cycles_per_etu - 1;

        subcarrier_states_in_bit = subcarrier_states_in_character( ...
            character_idx, subcarrier_states_start : ...
            subcarrier_states_end);

        bit_states_in_character(character_idx, bit_idx) = ...
            determine_bit_state(bit_idx, zero_state, ...
            subcarrier_states_in_bit);
    end

    %Check if all bit in one character are valid
    if any(bit_states_in_character(character_idx, 1 : 10) == -1)
        % The 11th column denotes if the whole character is valid or not
        bit_states_in_character(character_idx, 11) = -1;
```

```
    else
        bit_states_in_character(character_idx, 11) = 0;
    end

end


end

function bit_state = determine_bit_state(idx_of_bit_in_character, ...
    zero_state, subcarrier_states_in_bit)

% function bit_state = determine_bit_state(idx_of_bit_in_character, ...
%    zero_state, subcarrier_states_in_bit)
%
% DESCRIPTION: Function for determining the state of a bit from the
%              contained subcarrier states
%
%
%
% INPUT:
%       idx_of_bit_in_character.....Index if the current bit in the
%                                   current character
%       zero_state.....Subcarrier state corresponding to the bit state of
%                      logic 0
%       subcarrier_states_in_bit.....Matrix containing the subcarrier
%                                    states in each character as rows
%
% OUTPUT:
%       bit_state.....The state of the current bit.
%                     0 : Logic 0 (Data bit)
%                     1 : Logic 1 (Data bit)
%                     2 : Start bit
%                     3 : Stop bit
%                     -1: Invalid bit (e.g. not all subcarriers in bit
%                         have the same state or start bit is logic 1)
%

one_state = ~zero_state;

% Start bit
if idx_of_bit_in_character == 1
    if any(subcarrier_states_in_bit ~= zero_state)
        bit_state = -1;

    else
        bit_state = 2;
    end

    % Stop bit
elseif idx_of_bit_in_character == 10
    if any(subcarrier_states_in_bit ~= one_state)
        bit_state = -1;

    else
        bit_state = 3;
    end

    %Data bit
else
    % If all subcarriers in the bit are equal to the subcarrier state
    % corresponding to
    % logic one, the state of the bit is logic 1
    if all(subcarrier_states_in_bit == one_state)
        bit_state = 1;

        % The same but for logic 0
    elseif all(subcarrier_states_in_bit == zero_state)
        bit_state = 0;

        % Error case
```

```
    else
        bit_state = -1;
    end
end

end

function EOF_length_ok = check_EOF_length(EOF_start, ...
    phase_transitions_after_EOF_low, subcarrier_cycles_per_etu)

% function EOF_length_ok = check_EOF_length(EOF_start, ...
%    phase_transitions_after_EOF_low, subcarrier_cycles_per_etu)
%
% DESCRIPTION: Function for checking the lengh of the EOF sequence.
%
%              EOF consists of 10 to 11 etus of logic 0 and one rising
%              edge. The length of the low sequence is checked in this
%              function.
%
%
%
% INPUT:
%       EOF_start.....Start index of the EOF low sequence
%       phase_transitions_after_EOF_low.....Nx2 matrix containing the
%                                     indices and states after the EOF low
%                                     sequence (normally the this matrix
%                                     has only one row)
%       subcarrier_cycles_per_etu.....Subcarrier cycles in one etu
%
% OUTPUT:
%       EOF_length_ok.....Flag (0 or 1) indicating if the length of the
%                         EOF sequence is valid
%

EOF_length_ok = 1;
EOF_low_length = phase_transitions_after_EOF_low(1,1) - EOF_start + 1;

if EOF_low_length < 10 * subcarrier_cycles_per_etu
    warning(['EOF low length is %d subcarrier periods and therefore ', ...
        'is shorter than its minimum required length of 10 etus ', ...
        '(%d subcarrier periods).'], ...
        EOF_low_length, 10 * subcarrier_cycles_per_etu)
    EOF_length_ok = 0;

elseif EOF_low_length > 11 * subcarrier_cycles_per_etu
    warning(['EOF low length is %d subcarrier periods and therefore ', ...
        'is longer than its maximum allowed length of 11 etus ', ...
        '(%d subcarrier periods).'], ...
        EOF_low_length, 10 * subcarrier_cycles_per_etu)
    EOF_length_ok = 0;
end

end
```

N.12.24 envelope_postprocessing.m

```
function [env_up, env_lo, low_amplitude_subcarriers] = ...
    envelope_postprocessing(env_up, env_lo, ...
    subcarrier_indices, filter_length, max_idx, max_vals, min_idx, ...
    min_vals)
% function [env_up, env_lo, low_amplitude_subcarriers] = ...
%    envelope_postprocessing(env_up, env_lo, ...
%    subcarrier_indices, filter_length, max_idx, max_vals, min_idx, ...
%    min_vals)
%
% DESCRIPTION: Function for smoothing the input envelope with a moving
%   average filter of defined length
%
% INPUT:
%       env_up.....upper peak envelope of correlation signal [N x 1]
```

```
%       env_lo.....lower peak envelope of correlation signal [N x 1]
%       subcarrier_indices.....vector containing the indices of subcarrier
%           boundaries
%       filter_length...length of moving average filter
%       max_idx.....Indices of local maxima of correlation signal
%       max_vals.....Values of local maxima of correlation signal
%       min_idx.....Indices of local minima of correlation signal
%       min_vals.....Values of local minima of correlation signal
% OUTPUT:
%       env_up.....filtered upper peak envelope of correlation signal
%           [N x 1]
%       env_lo.....filtered lower peak envelope of correlation signal
%           [N x 1]
%       low_amplitude_subcarriers...binary vector containing '1' for
%           subcarrier with low amplitude

max_env = max(env_up);
min_env = min(env_lo);

samples_per_subcarrier = subcarrier_indices(2) - subcarrier_indices(1);

% An offset of 3/4 subcarriers results in one peak in each half subcarrier
% which allows simple subcarrier state detection
correlator_subcarrier_indices = subcarrier_indices - ...
    round( 3/4 * samples_per_subcarrier);

% ----- Smooth envelopes with moving average filter
env_up = envelope_smoothing(env_up, filter_length);
env_lo = envelope_smoothing(env_lo, filter_length);

% --- Due to correlator fade out set the last filter_length indices to the
% value of the envelope of index (filter_length - 1)
idx = subcarrier_indices(end) + (- filter_length - ...
    samples_per_subcarrier : 1 : 0);
env_up(idx) = env_up(idx(1) - 1);
env_lo(idx) = env_lo(idx(1) - 1);

% ----- Envelope modification in areas with low signal amplitude
%
% First, the local extrema in each subcarrier are determined. If the local
% maxima are below a defined threshold level AND the local minima are
% above another threshold level the subcarrier is marked as 'subcarrier
% with low amplitude' by writing a logic '1' to low_amplitude_subcarriers
% at the corresponding position.

N_subcarriers = length(subcarrier_indices) - 1;

low_amplitude_subcarriers = false(N_subcarriers, 1);

vals_for_mean_max = max_vals(max_vals > 0.75 * max(max_vals));
vals_for_mean_min = min_vals(min_vals < 0.75 * min(min_vals));

threshold_level_up = mean(vals_for_mean_max);
threshold_level_lo = mean(vals_for_mean_min);

% Empirically determined scaler for threshold level to address none ideal
% signals
threshold_scaler = 1 / 3;

for k = 1 : N_subcarriers
    subcarrier_start = correlator_subcarrier_indices(k);
    subcarrier_end = correlator_subcarrier_indices(k + 1);

    if k == 1 || k == N_subcarriers % First and last subcarrier peak have
        % only 50% of amplitude
        threshold_up = threshold_level_up * threshold_scaler * 1/2;
        threshold_lo = threshold_level_lo * threshold_scaler * 1/2;
    else
        threshold_up = threshold_level_up * threshold_scaler;
        threshold_lo = threshold_level_lo * threshold_scaler;
    end
```

```
    maxima_in_subcarrier = max_vals( max_idx >= subcarrier_start & ...
                                     max_idx <= subcarrier_end);

    minima_in_subcarrier = min_vals( min_idx >= subcarrier_start & ...
                                     min_idx <= subcarrier_end);

    if (~isempty(maxima_in_subcarrier) && mean(maxima_in_subcarrier) < ...
            threshold_up) || ...
       (~isempty(minima_in_subcarrier) && mean(minima_in_subcarrier) > ...
            threshold_lo)
        low_amplitude_subcarriers(k) = true;
    end
end

if any(low_amplitude_subcarriers)

    % Matrix containing all indices of all subcarriers with low amplitude
    % Second repmat() needed because matlab does not support matrix +
    % vector
    %
    % | 0 : samples_per_subcarrier -1 |    |low amplitude subcarrier 1|
    % | 0 : samples_per_subcarrier -1 |    |low amplitude subcarrier 2|
    % |  .                            | +  | .                        |
    % |  .                            |    | .                        |
    % |  .                            |    | .                        |
    % | 0 : samples_per_subcarrier -1 |    |low amplitude subcarrier k|
    %
    env_low_amplitude_idx = repmat(0 : samples_per_subcarrier - 1, ...
        length(find(low_amplitude_subcarriers)), 1) + ...
        ...
        repmat(correlator_subcarrier_indices(...
        low_amplitude_subcarriers == 1), 1, ...
        samples_per_subcarrier);


    env_up(env_low_amplitude_idx) = 1.1 * max_env;
    env_lo(env_low_amplitude_idx) = 1.1 * min_env;

end

% ----- Avoid detection of extrema before start of response
env_up(1 : correlator_subcarrier_indices(1)) = 1.1 * max_env;
env_up(correlator_subcarrier_indices(end) : end) = 1.1 * max_env;

env_lo(1 : correlator_subcarrier_indices(1)) = 1.1 * min_env;
env_lo(correlator_subcarrier_indices(end) : end) = 1.1 * min_env;


end

function env_filt = envelope_smoothing(env, filter_length)

% function env_filt = envelope_smoothing(env, filter_length)
%
% DESCRIPTION: Function for smoothing the input envelope with a moving
%   average filter of defined length
%
% INPUT:
%       env.....input envelope to be smoothed
%       filter_length.....Length of the moving average filter
% OUTPUT:
%       env_filt.....Filtered envelope


b = ones(1, filter_length) / filter_length;
group_delay = round(filter_length / 2);

% ----- Add backoff for filter fade-in and fade-out effects
N_backoff = 2 * filter_length;
```

```
env = [
    env(1) * ones(N_backoff, 1);
    env ; ...
    env(1) * ones(N_backoff, 1)
    ];

env_filt = filter(b, 1, env);

env_filt = env_filt(N_backoff + group_delay : end - N_backoff - 1 + ...
    group_delay);

end
```

N.12.25 LMA_over_time.m

```
function [usb_min, lsb_min, VLMA_max] = LMA_over_time(usb, lsb,  ...
    fft_step, subcarrier_indices, bit_states, SOF_start, encoding, ...
    ignore_last_subcarrier)

% function [usb_min, lsb_min, VLMA_max] = LMA_over_time(usb, lsb,  ...
%    fft_step, subcarrier_indices, bit_states, SOF_start, encoding, ...
%    ignore_last_subcarrier)
%
% DESCRIPTION: Function to extract the relevant section of LMA
%              computed in sliding_lma(); Optionally the LMA can be
%              plotted
%
% INPUT:
%      usb.....Complex vector containing the values of the upper sideband
%                of the sliding lma
%      lsb.....Complex vector containing the values of the lower sideband
%                of the sliding lma
%      fft_step.....Step width of the sliding lma
%      subcarrier_indices.....Vector containing the indices of the
%                             subcarrier boundaries
%      bit_states.....Bit states (relevant for Type A 106kBps signals)
%      SOF_start.....Start of SOF (relevant for Type B signals)
%      encoding.....Encoding of the PICC response
%      ignore_last_subcarrier.....Flag indicating that the last subcarrier
%                                 in the frame should be ignored

ylabel_txt = 'Sideband amplitudes in mV(p)';
xlabel_txt = 'Subcarrier periods';
PLOT = 0;

if strcmp(encoding.interface_type, 'Type_A')
    %Type A, 106kBps
    if strcmp(encoding.bitrate, '106')
        if ignore_last_subcarrier
            bit_states = bit_states(1 : end - 1);
        end

        zero_one_state_transitions = find(diff(bit_states) == 1);
        lma_start_end_matrix = [(zero_one_state_transitions - 1) * 8 + ...
            5, (zero_one_state_transitions - 1) * 8 + 7];

        lma_start_end_subcarrier_idx = ...
            subcarrier_indices(lma_start_end_matrix);

        % Behaviour of software-packages:
        % In case of lma_start_end_subcarrier_idx is a 1x2 vector (only
        % 0 to 1 bit state transition found) the uppper instruction
        % results in a 2x1 vector and not in a 1x2 vector as expected.
        % The if-clause below is added to treat this case.
        if size(lma_start_end_subcarrier_idx) == [2,1]
            lma_start_end_subcarrier_idx = lma_start_end_subcarrier_idx';
        end

        lma_start_end_idx = [ceil(lma_start_end_subcarrier_idx(:, 1) / ...
            fft_step), ceil(lma_start_end_subcarrier_idx(:, 2) / ...
```

```
            fft_step)];

        % Absolute values in mV
        abs_usb = abs(usb);
        abs_lsb = abs(lsb);

        sig_usb = zeros(size(lma_start_end_idx,1),3);
        sig_lsb = zeros(size(lma_start_end_idx,1),3);

        for k = 1 : size(lma_start_end_idx,1)
            lma_range = lma_start_end_idx(k, 1) : lma_start_end_idx(k, 2);

             sig_usb(k,:) = abs_usb(lma_range);
             sig_lsb(k,:) = abs_lsb(lma_range);
        end
        sig_usb = sig_usb(:);
        sig_lsb = sig_lsb(:);

        title_txt = 'USB and LSB during 8 subcarrier periods occurrences';

    %Type A, > 106 kBps
    else
        start_idx = ceil(subcarrier_indices(1) / fft_step);
        end_idx = floor(subcarrier_indices(26) / fft_step);

        sig_usb = abs(usb(start_idx : end_idx)).';
        sig_lsb = abs(lsb(start_idx : end_idx)).';

        title_txt = 'USB and LSB in SOC';
    end

%Type B
else
    start_idx = ceil(subcarrier_indices(1) / fft_step);
    end_idx = floor(subcarrier_indices(SOF_start - 6) / fft_step);

    sig_usb = abs(usb(start_idx : end_idx)).';
    sig_lsb = abs(lsb(start_idx : end_idx)).';

    title_txt = 'USB and LSB in TR1';
end
    if PLOT
        figure()
        p1 = plot( sig_usb );
        hold on
        p2 = plot( sig_lsb );
        legend([p1, p2], {'USB', 'LSB'})
        xlabel(xlabel_txt)
        ylabel(ylabel_txt)
        title(title_txt)
        axis tight
        grid minor
        set(gca, 'box', 'on')
    end
    usb_min = min(sig_usb);
    lsb_min = min(sig_lsb);
    VLMA_max = max(mean([sig_usb, sig_lsb], 2));
end
```

Annex O
(normative)

Conformance test plan

O.1 PCD conformance test plan

O.1.1 Test command sequences

O.1.1.1 General

This subclause defines the test command sequences used for tests of ISO/IEC 14443-1, ISO/IEC 14443-2, ISO/IEC 14443-3 and ISO/IEC 14443-4 parameters, as correspondingly indicated in Table O.8, Table O.9, Table O.11 and Table O.12.

O.1.1.2 Examples of test commands

O.1.1.2.1 UT_TEST_COMMAND1

UT_TEST_COMMAND1 as defined in Table O.1 and Table O.2 recommends the instruction coding used as the default instruction for scenarios not needing PCD chaining.

Table O.1 — UT_TEST_COMMAND1, command UT_APDU definition

Command	COMMAND UT_APDU
UT_TEST_COMMAND1	'00 DA 00 00 0E AA AA F0 FF 04 05 06 07 08 09 0A 0B 0C 0D'[a]
[a] The pattern 'AA AA F0 FF' eases the application of Annex E.	

Table O.2 — UT_TEST_COMMAND1, response UT_APDU definition

Answer	RESPONSE UT_APDU
Answer to UT_TEST_COMMAND1	'01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 90 00'

O.1.1.2.2 UT_TEST_COMMAND2

UT_TEST_COMMAND2 as defined in Table O.3 and Table O.4 recommends the instruction coding used as the default instruction for scenarios dealing with PCD chaining.

Table O.3 — UT_TEST_COMMAND2, command UT_APDU definition

Command	COMMAND UT_APDU
UT_TEST_COMMAND2	For FSCI ≤ 8: '00 DA 00 00 FF 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11...FE FF' For FSCI > 8: '00 DA 00 00 00 10 00 00 01 00 02 00 03 00 04 00 05 ...0F FF 10 00'

Table O.4 — UT_TEST_COMMAND2, response UT_APDU definition

Answer	RESPONSE UT_APDU
Answer to UT_TEST_COMMAND2	First I-block: '01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F' Second I-block: '11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F' Third I-block: '90 00'

O.1.1.3 Default test command sequence

The test command sequence as defined in Table O.5 shall be applied as default test command sequence for procedures or scenarios dealing with at least one I-block.

Table O.5 — Test command sequence (at least one I-blocks)

Step	Command
1	INITIALIZE_PCD_TEST_MODE
2	INITIATE_ANTICOLLISION
3	SEND_UT_APDU (UT_TEST_COMMAND1)

O.1.1.4 Test_command_sequence_1

The test command sequence as defined in Table O.6 shall be applied for scenarios not needing PCD chaining and exchanging two I-blocks.

Table O.6 — Test command sequence (no PCD chaining, two I-blocks)

Step	Command
1	INITIALIZE_PCD_TEST_MODE
2	INITIATE_ANTICOLLISION
3	SEND_UT_APDU (UT_TEST_COMMAND1)
4	SEND_UT_APDU (UT_TEST_COMMAND1)

O.1.1.5 Test_command_sequence_2

The test command sequence as defined in Table O.7 shall be applied for scenarios dealing with PCD chaining.

Table O.7 — Test command sequence (PCD chaining)

Step	Command
1	INITIALIZE_PCD_TEST_MODE
2	INITIATE_ANTICOLLISION
3	SEND_UT_APDU (UT_TEST_COMMAND2)
NOTE UT_TEST_COMMAND1 may be sent before step 3.	

O.1.2 Test of ISO/IEC 14443-1 PCD parameters

A PCD is compliant with ISO/IEC 14443-1 if its test result is PASS for all applicable tests listed in Table O.8.

Table O.8 — Test of ISO/IEC 14443-1 PCD parameters

Test	Reference	Test result PASS or FAIL or N/A	Comments
Alternating magnetic field	6.1.1		Test is performed after step INITIALIZE_PCD_TEST_MODE.

O.1.3 Test of ISO/IEC 14443-2 PCD parameters

A PCD is compliant with ISO/IEC 14443-2 if its test result is PASS for all applicable tests listed in Table O.9.

Table O.9 — Test of ISO/IEC 14443-2 PCD parameters

Test	Reference	Specific test conditions	Test command sequence	Test result PASS or FAIL or N/A	Comments
Carrier frequency f_c as defined in ISO/IEC 14443-2:2020, 6.2					Use appropriate frequency meter to measure the frequency when the operating field is present.
PCD field strength	7.1.1				Test is performed after step INITIALIZE_PCD_TEST_MODE.
Modulation index m and waveform	7.1.4	Reference PICCs resonance frequencies values other than the value defined in Table 4 are optional.	For a PCD to PICC bit rate of $f_c/128$, any activation command. For PCD to PICC bit rates higher than $f_c/128$, any ISO/IEC 14443-4 frame from default test command sequence (see O.1.1.3), after appropriate PCD to PICC bit rate selection.		Testing with other Reference PICCs resonance frequencies increases test coverage.
Load modulation reception for PICC to PCD bit rates of $f_c/128$, $f_c/64$, $f_c/32$ and $f_c/16$	7.1.6		For a PICC to PCD bit rate of $f_c/128$, any activation command. For PICC to PCD bit rates higher than $f_c/128$, any I-block from default test command sequence (see O.1.1.3), after appropriate PICC to PCD bit rate selection.		

Table O.9 *(continued)*

Test	Reference	Specific test conditions	Test command sequence	Test result PASS or FAIL or N/A	Comments
Load modulation reception for PICC to PCD bit rates of $f_c/8$, $f_c/4$ and $f_c/2$	7.1.7	Reference PICCs resonance frequencies values other than the value defined in Table 4 are optional.	Any I-block from default test command sequence (see O.1.1.3), after appropriate PICC to PCD bit rate selection.		Testing with other Reference PICCs resonance frequencies increases test coverage.
PCD EMD immunity test	7.1.8		All ISO/IEC 14443-3 and ISO/IEC 14443-4 commands of default test command sequence (see O.1.1.3).		

O.1.4 Test of ISO/IEC 14443-3 PCD parameters

The test conditions defined in Table O.10 shall be applied.

Table O.10 — ISO/IEC 14443-3 PCD parameters test conditions

Conditions	Values
Type	Type A and Type B
Test position	Any position in the operating volume providing reliable communication.
Reference PICC	Reference PICC 1

A PCD is compliant with ISO/IEC 14443-3 if its test result is PASS for all applicable tests listed in Table O.11.

Table O.11 — Test of ISO/IEC 14443-3 PCD parameters

Test	Reference	Test command sequence	Test result PASS or FAIL or N/A
Measuring timing	H.1.7		
PCD EMD recovery test	8.1.1	All commands used for default test command sequence (see O.1.1.3).	
Frame Delay Time PICC to PCD	H.2.1	Test_command_sequence_1 (see O.1.1.4)	See H.6
Request Guard Time	H.2.2	INITIALIZE_PCD_TEST_MODE	See H.6
Handling of bit collision during ATQA	H.2.3	INITIALIZE_PCD_TEST_MODE	See H.6
Handling of anticollision loop	H.2.4	INITIATE_ANTICOLLISION	See H.6
Handling of RATS and ATS	H.2.5	Default test command sequence (see O.1.1.3)	See H.6
Frame size selection mechanism	H.2.7	Test_command_sequence_2 (see O.1.1.5)	See H.6
Handling of Start-up Frame Guard Time	H.2.8	Default test command sequence (see O.1.1.3)	See H.6

Table O.11 *(continued)*

Test	Reference	Test command sequence	Test result PASS or FAIL or N/A
Handling of the CID during activation by the PCD	H.2.9	Default test command sequence (see O.1.1.3)	See H.6
Handling of parity bit	H.2.10	Default test command sequence (see O.1.1.3)	See H.6
Frame size selection mechanism	H.3.2	Test_command_sequence_2 (see O.1.1.5)	See H.6
Handling of the CID during activation by the PCD	H.3.3	Default test command sequence (see O.1.1.3)	See H.6
Frame Delay Time PICC to PCD (TR2)	H.3.4	Test_command_sequence_1 (see O.1.1.4)	See H.6
Handling of Start-up Frame Guard Time	H.3.5	Default test command sequence (see O.1.1.3)	See H.6
Type B PCD framing tests	H.3.6	Default test command sequence (see O.1.1.3)	See H.6
Handling of the polling loop	H.4.2	INITIATE_ANTICOLLISION	See H.6
Continuous monitoring of packets sent by the PCD	H.5		See H.6

O.1.5 Test of ISO/IEC 14443-4 PCD parameters

The test conditions defined in Table O.10 shall be applied.

A PCD is compliant with ISO/IEC 14443-4 if its test result is PASS for all applicable tests listed in Table O.12.

Table O.12 — Test of ISO/IEC 14443-4 PCD parameters

Test	Reference	Test command sequence	Test result PASS or FAIL or N/A
Reaction of the PCD to request for waiting time extension	H.4.3	Test_command_sequence_1 (see O.1.1.4)	See H.6
Error detection and recovery	H.4.4	For procedures 9 and 10, use Test_command_sequence_2 (see O.1.1.5). For all others procedures, use default test command sequence (see O.1.1.3).	See H.6
Handling of NAD during chaining	H.4.5	Test_command_sequence_2 (see O.1.1.5)	See H.6
Continuous monitoring of packets sent by the PCD	H.5		See H.6
High bit rate selection	Annex I	Default test command sequence (see O.1.1.3)	
Frame format selection	L.1.1		
Error correction mechanism	L.2.1		

O.2 PICC conformance test plan

O.2.1 Test command sequences

This subclause defines the test command sequences used for tests of ISO/IEC 14443-1, ISO/IEC 14443-2, ISO/IEC 14443-3 and ISO/IEC 14443-4 parameters, as correspondingly indicated in Table O.13, Table O.14, Table O.16 and Table O.17.

The definition of the applied sequences and commands TEST_COMMAND_SEQUENCE1, TEST_COMMAND1, TEST_COMMAND2, TEST_COMMAND3 and TEST_COMMAND4 (see 3.2) depends on the applicative layer and shall be provided by the applicant. Any test command sequence shall be sent after the PICC activation process described in G.5.1.2, including the additional sequences which may be required by the PICC to be ready for the test, e.g., a bit rates selection sequence.

O.2.2 Test of ISO/IEC 14443-1 PICC parameters

A PICC is compliant with ISO/IEC 14443-1 if its test result is PASS for all applicable tests listed in Table O.13.

Table O.13 — Test of ISO/IEC 14443-1 PICC parameters

Test	Reference	Specific test conditions	Test result PASS or FAIL or N/A	Comments
Antenna size and PICC class verification as defined in ISO/IEC 14443-1:2018, Annex A				This test is not applicable if no PICC class is declared in Table 6. The applied method is under responsibility of the test laboratory.
Alternating magnetic field	6.2.1			

O.2.3 Test of ISO/IEC 14443-2 PICC parameters

A PICC is compliant with ISO/IEC 14443-2 if its test result is PASS for all applicable tests listed in Table O.14.

Table O.14 — Test of ISO/IEC 14443-2 PICC parameters

Test	Reference	Specific test conditions	Test command sequence	Test result PASS or FAIL or N/A	Comments
PICC transmission	7.2.1	Any PCD to PICC bit rate may be used.	For a PICC to PCD bit rate of f_c/128, REQA or REQB. For PICC to PCD bit rates higher than f_c/128, TEST_COMMAND4 after appropriate PICC to PCD bit rate selection.		Different values of f_c are used in the test method.
PICC EMD level and low EMD time test	7.2.2	Any PCD to PICC bit rate may be used.	All ISO/IEC 14443-3 commands and TEST_COMMAND_SEQUENCE1 after appropriate PICC to PCD bit rate selection.		

Table O.14 *(continued)*

Test	Reference	Specific test conditions	Test command sequence	Test result PASS or FAIL or N/A	Comments
PICC reception	7.2.3	Any PICC to PCD bit rate may be used.	For PCD to PICC bit rates higher than f_c/128, TEST_COMMAND1(1) after appropriate PCD to PICC bit rate selection.		
PICC resonance frequency	7.2.4				This test is not applicable if no resonance frequency range is declared in Table 6.
PICC maximum loading effect	7.2.5				
PICC operating field strength	7.2.6	Any PICC to PCD bit rate may be used.	TEST_COMMAND_SEQUENCE1 after appropriate PCD to PICC bit rate selection.		

O.2.4 Test of ISO/IEC 14443-3 PICC parameters

The test conditions defined in Table O.15 shall be applied.

Table O.15 — ISO/IEC 14443-3 PICC parameters test conditions

Conditions	Values
Field strength	Any value between H_{min} and H_{max}

A PICC is compliant with ISO/IEC 14443-3 if its test result is PASS for all applicable tests listed in Table O.16.

Table O.16 — Test of ISO/IEC 14443-3 PICC parameters

Test	Reference	Test result PASS or FAIL or N/A	Comments
Measuring timing	G.1.7		Continuous monitoring of packets sent by the PICC.
Polling	G.3.2		
Testing of the PICC Type A state transitions	G.3.3		Some state transitions are not applicable to PICCs not compliant with ISO/IEC 14443-4.
Handling of Type A anti-collision	G.3.4		
Handling of Frame Delay Time PICC to PCD and SFGT	G.3.8		This test is not applicable to PICCs not compliant with ISO/IEC 14443-4.
Polling	G.4.2		
PICC framing and bit rates capability	G.4.3		
Testing of the PICC Type B state transitions	G.4.4		
Handling of Type B anti-collision	G.4.5		

Table O.16 *(continued)*

Test	Reference	Test result PASS or FAIL or N/A	Comments
Handling of ATTRIB	G.4.6		
Handling of Maximum Frame Size	G.4.7		For some Maximum Frame Size Code in ATTRIB values, the PICC may never chain. In such case, this test is not applicable for these values. This test is not applicable to PICCs not compliant with ISO/IEC 14443-4.
Handling of TR2 and SFGT	G.4.8		Some steps are not applicable to PICCs not compliant with ISO/IEC 14443-4.
PICC supporting Type A and Type B	G.5.7		

O.2.5 Test of ISO/IEC 14443-4 PICC parameters

The test conditions defined in Table O.15 shall be applied.

A PICC is compliant with ISO/IEC 14443-4 if its test result is PASS for all applicable tests listed in Table O.17.

Table O.17 — Test of ISO/IEC 14443-4 PICC parameters

Test	Reference	Test result PASS or FAIL or N/A	Comments
Handling of PPS request	G.3.6		
Handling of FSD	G.3.7		For some FSDI values, the PICC may never chain. In that case, this test is not applicable for these values.
PICC bit rates capability	G.3.9		
PICC reaction to ISO/IEC 14443-4 scenarios	G.5.2		
Handling of PICC error detection	G.5.3		
PICC reaction on CID	G.5.4		
PICC reaction on NAD	G.5.5		
PICC reaction on S(PARAMETERS) blocks	G.5.6		
RFU fields and values	G.1.6		Continuous monitoring of packets sent by the PICC.
Frame format selection	L.1.2		
Error correction mechanism	L.2.2		
PICC reaction to ISO/IEC 14443-4 scenarios when frame with error correction is activated	L.3.1		
Reduction of bit rate during chaining when frame with error correction is activated	L.3.2		

Annex P
(normative)

PICC Type A Frame Delay Time (FDT) determination method

P.1 Overview

The working principle of the PICC Type A Frame Delay Time (FDT) determination method is illustrated in Figure P.1.

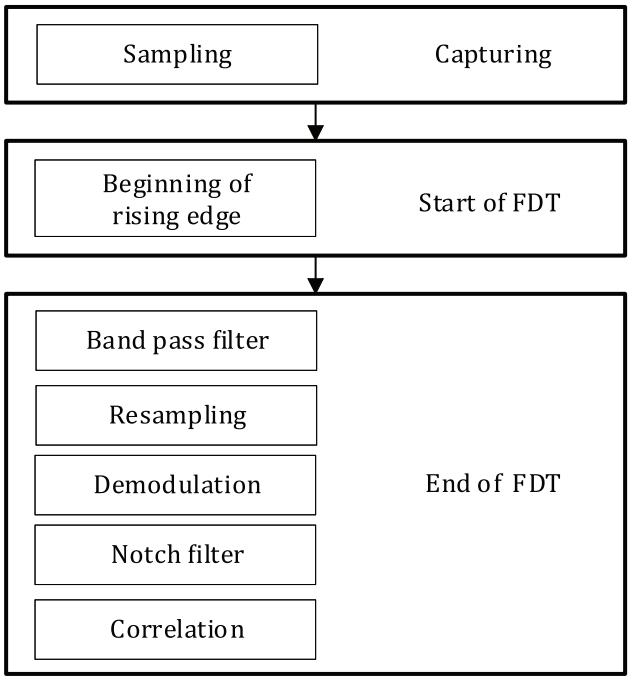


Figure P.1 — FDT determination method block diagram

Each block is described in following clauses.

NOTE The intrinsic uncertainty of the method is ± 0,5/f_c.

P.2 Capturing

The digital sampling device/oscilloscope used for signal capturing shall fulfill the requirements defined in 5.2.

The time and voltage data as captured by both the calibration coil and the load modulation test circuit (also known as sense coils), with at least 128 carrier periods before the last pause transmitted by the PCD and at least 128 carrier periods after the PICC start of communication shall be transferred to a suitable computer. Both, the calibration coil and the load modulation test circuit signals, shall be captured by the same digital sampling device as illustrated in Figure P.2.

The signal processing afterwards shall use the actual 1/f_c derived from the calibration coil signal as time basis.

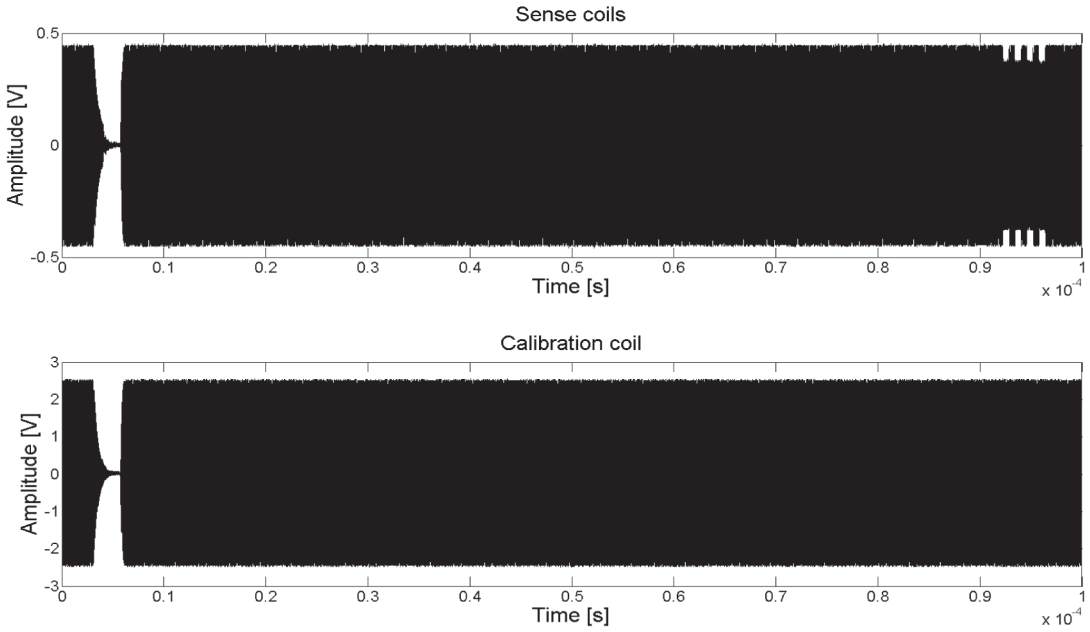


Figure P.2 — Example signals captured for FDT determination

P.3 Determination of the Start of FDT

The captured calibration coil signal shall be used to determine the beginning of the rising edge of the last pause transmitted by the PCD. The method described in Annex E shall be used.

P.4 Determination of the End of FDT

P.4.1 General

The captured calibration coil and sense coils signals shall be processed as described in the following clauses.

P.4.2 Band pass filter

The calibration coil and sense coils signals shall be filtered as specified in E.3.1.

P.4.3 Resampling

The filtered calibration coil and sense coils signals shall be resampled to an integer number multiple of f_c using linear interpolation. The integer number shall be at least 37.

P.4.4 Demodulation

The resampled sense coils signal shall be demodulated by the resampled calibration coil signal, to which phase shifts of 0°, 45°, 90° and 135° shall be applied, as shown in Figure P.3.

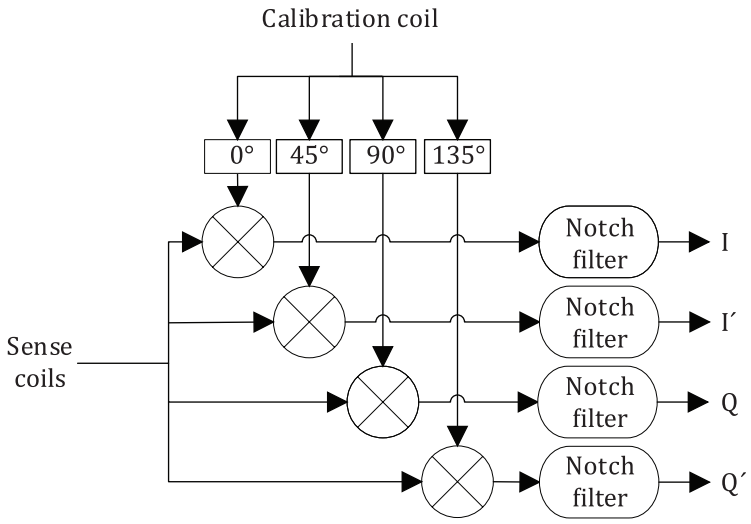


Figure P.3 — Demodulation scheme

P.4.5 Notch filter

The filter as specified in E.5.1 shall be applied to the demodulated signals in the following way:

1) After filtering the demodulated signals in forward direction, the filtered signals shall be reversed and filtered again.

2) After second filtering, the signals shall be reversed again to obtain signals I, I′, Q and Q′ (see Figure P.3).

P.4.6 Correlation

Each of the signals I, I′, Q and Q′ shall be cross-correlated with the signal shown in Figure P.4. From both the positive correlation maximum and negative correlation minimum of all correlation results, the later occurrence represents the center of the PICC start of communication. From this moment, $64/f_c$ shall be subtracted to get the first modulation edge of the PICC response.

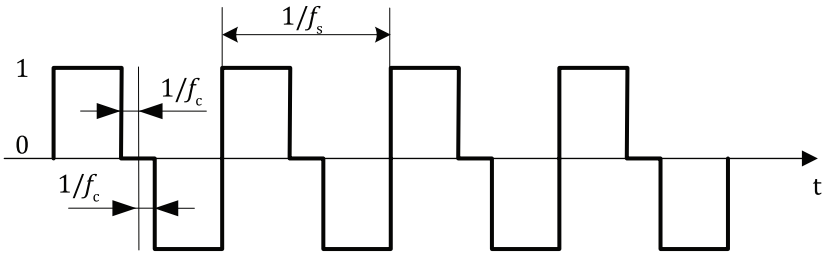


Figure P.4 — Correlation signal

P.5 Program of the FDT determination method

P.5.1 General

The following program written in the high-level interpreted programming language GNU Octave gives an example for the implementation of the FDT determination method. Alternatively this code program may be executed using MATLAB®.

P.5.2 fdt.m

```
function [ FDT ] = fdt( bitrate, t , sig )

% framedelaytime_in_fc = fdt( samplingtime vector,  [SenseCoil CalibrationCoil])

close all
debug          =  0;   %debug =  1; %debug =  0;
FDT.fdt_fc     =  0;
FDT.passfail   = 'NOT_FIN';
FDT.error      = 'NO_ERROR';

fs             = floor(1/(t(2)-t(1)));
param.minisofs = 500e6;     %at least 500 million samples per second sampling frequency,
as specified in 5.2
param.f_down   = 13.56e6*37;
fc_nom         = 13.56e6;

if fs < param.minisofs
     if debug == 1
          fprintf('Warning: Sampling frequency too low (see ISO/IEC 10373-6, 5.2
Oscilloscope) Fs =%d!\n\n',fs)
     end
      FDT.error = 'Warning: Sampling frequency too low (see ISO/IEC 10373-6, 5.2
Oscilloscope)';
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% 1. LP 100 MHz - Filter before downsampling

cutOff_freq = 100E6;
if fs <= cutOff_freq*2;
      if debug == 1
          fprintf('Error: Sampling frequency too low (see ISO/IEC 10373-6, 5.2
Oscilloscope) Fs =%d!\n\n',fs)
      end
      FDT.error = 'Error: Sampling frequency too low (see ISO/IEC 10373-6, 5.2
Oscilloscope)';
      return;
end
% perform LP -filtering
Wn       = [ cutOff_freq]/(fs/2);
[b,a]    = butter(2, Wn);
sigtemp  = sig;
sig      = filtfilt(b,a,sig);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%2. re-sampling   Fs new

t2       =linspace(t(1),t(end),(t(end)-t(1))*param.f_down);
sig      =interp1(t,sig,t2);
sigtemp  =interp1(t,sigtemp,t2);
t        =t2;
fs       =param.f_down;
samples_per_carrier_nom = round(fs/fc_nom);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%3 Check Signal same length

cal_coil_sig  = sig(:,2);
hhb_sig       = sig(:,1);

N_cc          = length(cal_coil_sig);
N_hhb         = length(hhb_sig);

if N_cc > N_hhb
    cal_coil_sig = cal_coil_sig( 1 : N_hhb );
    t=t(1 : N_hhb);
```

```
else
    hhb_sig = hhb_sig( 1 : N_cc );
    t=t(1 : N_hhb);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% 5. BP - Filter   13,56MHZ +/- 10MHz

BW       = 10E6;
Wn       = [fc_nom-BW, fc_nom+BW]/(fs/2);
[b,a]    = butter(2, Wn);
cal_coil_sig  = filtfilt(b,a, cal_coil_sig);
hhb_sig       = filtfilt(b,a, hhb_sig);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% 6. find end of t1 of rising edge

hilbert_cal_coil_sig = hilbert(cal_coil_sig);
bb_hilbert           = abs(hilbert_cal_coil_sig);

N_filt               = samples_per_carrier_nom; %
IQFilt               = ones(1, N_filt)/N_filt; % define length of moving average filter
bb_hilbert_filt      = filtfilt(IQFilt,1,bb_hilbert);

[dist,Valu]   = hist(bb_hilbert_filt,5000);
[Y,index]     = max(dist);
Hmax          = Valu(index);

switch bitrate
    case '128/fc',
         [Y,index]=find(bb_hilbert_filt(samples_per_carrier_nom*16:end-samples_per_
carrier_nom*1172) < 0.05*Hmax);
    case {'64/fc' '32/fc' '16/fc'}
         a          = min(bb_hilbert_filt(samples_per_carrier_nom*16:end-samples_per_
carrier_nom*1172))/Hmax;
         [Y,index]  = find(bb_hilbert_filt(samples_per_carrier_nom*16:end-samples_per_
carrier_nom*1172) < (a+0.1*(1-a))*Hmax);
end
    if isempty(Y)
          if debug == 1
              fprintf('Error: No Modulation (EOF) detected \n\n');
          end
             FDT.error = 'Error: No Modulation (EOF) detected';
         return
    end
indexeof=Y(end)+samples_per_carrier_nom*16-1;

t_fc      = (unwrap(angle(hilbert_cal_coil_sig)))/(2*pi);
FDTstart  = t_fc(indexeof);
fc        = (t_fc(indexeof+samples_per_carrier_nom*(20+128))-t_fc(indexeof+samples_per_
carrier_nom*20))/128*13.56e6;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

switch bitrate
     case '128/fc',
         % measure t4 timing
         [Y,index]=find(bb_hilbert_filt(indexeof:end-samples_per_carrier_nom*1172-7) <
0.6*Hmax);

         indext4=Y(end)+indexeof-1;
         t4=t(indext4)-t(indexeof);
         t_fc=(t_fc-t_fc(indext4)+t4*fc);

      case {'64/fc' '32/fc' '16/fc'}
          % measure t6 timing
         [Y,index]=find(bb_hilbert_filt(indexeof:end-samples_per_carrier_nom*1172-7) <
(a+0.9*(1-a))*Hmax);
```

```
        indext6=Y(end)+indexeof-1;
        t6=t(indext6)-t(indexeof);
        t_fc=(t_fc-t_fc(indext6)+t6*fc);

end


if debug == 1
          figure(1)
          subplot(3,1,1)
          hold on
          plot(t_fc,bb_hilbert_filt,'m--')
          plot(t_fc,cal_coil_sig,'b')
          line=[-1:0.001:1];
          plot(0,line,'r','linewidth',1)
          axis([-10 30 -Hmax*1.2 Hmax*1.2])
          xlabel('time [1/fc]')
          ylabel('Calibration coil envelope')
          grid on
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% 6. Homodyne Demodulation

% make demodulation signal (0° shifted version)
cal_coil_sig=cal_coil_sig/(max(abs(cal_coil_sig(round(end/2:end/2 + samples_per_carrier_
nom*16)))));

% make demodulation signal (90° shifted version)
cal_coil_90 = -imag(hilbert(cal_coil_sig));
cal_coil_90 = cal_coil_90/(max(abs(cal_coil_90(round(end/2:end/2 + samples_per_carrier_
nom*16)))));

% make demodulation signal (45° shifted version)
cal_coil_45 = cal_coil_90+cal_coil_sig;
cal_coil_45 = cal_coil_45/(max(abs(cal_coil_45(round(end/2:end/2 + samples_per_carrier_
nom*16)))));

% make demodulation signal (135° shifted version)
cal_coil_135 = cal_coil_90-cal_coil_sig;
cal_coil_135 = cal_coil_135/(max(abs(cal_coil_135(round(end/2:end/2 + samples_per_carrier_
nom*16)))));

bb_sig(:,1) = hhb_sig.*cal_coil_sig;
bb_sig(:,2) = hhb_sig.*cal_coil_45;
bb_sig(:,3) = hhb_sig.*cal_coil_90;
bb_sig(:,4) = hhb_sig.*cal_coil_135;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% 7 AM - Filtering to suppress second harmonic

N_filt          = ceil(fs/fc); %
IQFilt          = ones(1, N_filt)/N_filt; % define length of moveing average filter
bb_sig_filt     = filtfilt(IQFilt,1,(bb_sig));
bb_hilbert_filt = filtfilt(IQFilt,1,bb_hilbert);


clear bb_sig; % clean up
clear bb_hilbert; % clean up

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% 8 Correlation signal--> length  4 Fsub
fc_fs = fc/fs;

correlation_sig=[ones(1,6*round(fs/fc)) zeros(1,4*round(fs/fc)) -ones(1,6*round(fs/fc))];
correlation_sig= [correlation_sig correlation_sig correlation_sig correlation_sig];

% **Correlation signal--> length of 4 Fsub
```

```
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++Convolution/Faltung

faltung(:,1) = (conv(bb_sig_filt(:,1),correlation_sig));
faltung(:,2) = (conv(bb_sig_filt(:,2),correlation_sig));
faltung(:,3) = (conv(bb_sig_filt(:,3),correlation_sig));
faltung(:,4) = (conv(bb_sig_filt(:,4),correlation_sig));
faltung      = faltung(1:end-length(correlation_sig)+1,:);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Find Response first edge (FDT)%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%suppress  "deaf time"%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

deaftime = samples_per_carrier_nom*(1172-32);
faltung(1:indexeof+deaftime,:)  = faltung(1:indexeof+deaftime,:)*0;

% find strongest signal
[Y,index]   = max(max(faltung));
indexstrong = index;
Ystrong=Y;
faltung     = faltung(:,index);
bb_sig_filt = bb_sig_filt(:,index);

[Y,index]   = find(faltung > Ystrong/2);
faltung(Y(1)+ samples_per_carrier_nom*64 : end )  =  faltung(Y(1)+samples_per_carrier_
nom*64:end) *0;

if debug == 1
        figure(1)
           subplot(3,1,2)
        plot(t_fc,(faltung)/max(abs(faltung)),'r')
        hold on
        ylabel('correlation')
        xlabel('time [1/fc]')
        grid on
end

[Y(1),index(1)] = max(faltung);
[Y(2),index(2)] = min(faltung);

indexsof        = max(index);
FDTend=t_fc(indexsof);

     if 10 > max(faltung)/max(faltung( ((indexsof-samples_per_carrier_nom*96)  :
(indexsof-samples_per_carrier_nom*64)) ))
         if debug == 1
           fprintf('Warning: No Modulation (SOF) detected \n\n');
         end
           FDT.error = 'Warning: No Modulation (SOF) detected';
     end

% 9 PASS/FAIL     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

switch bitrate
    case '128/fc'
    FDT.fdt_fc=t_fc(indexsof)-t_fc(indext4)+((indext4-indexeof)*fc_fs)-64;
    Limit_min_0=1172;
    Limit_max_0=1177.424;
    Limit_min_1=1236;
    Limit_max_1=1241.424;

    case '64/fc'
    FDT.fdt_fc=t_fc(indexsof)-t_fc(indexeof)-64;
    rest=mod(FDT.fdt_fc,128);
    Limit_min_0=FDT.fdt_fc-rest+116;
    Limit_max_0=FDT.fdt_fc-rest+116+5.4239;
    Limit_min_1=FDT.fdt_fc-rest+20;
    Limit_max_1=FDT.fdt_fc-rest+20+5.4239;
```

```
    case '32/fc'
    FDT.fdt_fc=t_fc(indexsof)-t_fc(indexeof)-64;
    rest=mod(FDT.fdt_fc,128);
    Limit_min_0=FDT.fdt_fc-rest+100;
    Limit_max_0=FDT.fdt_fc-rest+100+5.4239;
    Limit_min_1=FDT.fdt_fc-rest+116;
    Limit_max_1=FDT.fdt_fc-rest+116+5.4239;

    case '16/fc'
    FDT.fdt_fc=t_fc(indexsof)-t_fc(indexeof)-64;
    rest=mod(FDT.fdt_fc,128);
    Limit_min_0=FDT.fdt_fc-rest+92;
    Limit_max_0=FDT.fdt_fc-rest+92+5.4239;
    Limit_min_1=FDT.fdt_fc-rest+100;
    Limit_max_1=FDT.fdt_fc-rest+100+5.4239;

end
   if  (Limit_min_0-0.5 < FDT.fdt_fc && FDT.fdt_fc < Limit_max_0+0.5)
           FDT.passfail='PASS if Command last bit =(0)b';
           elseif (Limit_min_1-0.5 < FDT.fdt_fc && FDT.fdt_fc < Limit_max_1+0.5)
                   FDT.passfail='PASS if Command last bit =(1)b';
                  else
                        FDT.passfail='FAIL';
    end

if debug == 1

       figure(1)
           subplot(3,1,2)
       hold on
       line=[-1:0.01:1];
       plot(FDTend,line,'g')
            subplot(3,1,3)
       hold on

       plot(t_fc,hhb_sig/max(abs(hhb_sig)),'g')
       plot(t_fc,sigtemp(:,1)/max(abs(sigtemp(:,1))),'g--')
       plot(t_fc,bb_sig_filt/max(abs(bb_sig_filt)) ,'r')
       plot(FDTend-64,line,'r','LineWidth',0.7)

       plot(Limit_min_0,line,'b','LineWidth',0.7)
       plot(Limit_max_0,line,'b','LineWidth',0.7)
       plot(Limit_min_1,line,'b','LineWidth',0.7)
       plot(Limit_max_1,line,'b','LineWidth',0.7)
       xlabel('time [1/fc]')
       ylabel('Sense coil')
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

end
```

P.5.3 main_FDT_tool.m

```
clear all
close all
clc

pkg load signal % needed for the function "butter" octave

[FileName, PathName] = uigetfile('*.csv','Select File','MultiSelect','off');
FileLocation = [PathName,FileName];
fid = fopen(FileLocation);

[Data] = textscan(fid, '%f %f %f', 'headerlines', 1, 'delimiter', ',');
t   = Data{1};
sig = [Data{2}, Data{3}];

bitrate= '128/fc';
%bitrate= '64/fc';
```

```
%bitrate= '32/fc';
%bitrate= '16/fc';

% framedelaytime = fdt(PCD-PICC-bitrate, samlingtime Vector,  [SenseCoil
CalibrationCoil]);
FDT=fdt(bitrate,t,sig)
```

Bibliography

[1] ISO/IEC Guide 98-3, *Uncertainty of measurement — Part 3: Guide to the expression of uncertainty in measurement (GUM:1995)*

[2] ISO/IEC 9646-1, *Information technology — Open Systems Interconnection — Conformance testing methodology and framework — Part 1: General concepts*

[3] ISO/IEC 9646-2, *Information technology — Open Systems Interconnection — Conformance testing methodology and framework — Part 2: Abstract Test Suite specification*

[4] ISO/IEC 9646-3, *Information technology — Open Systems Interconnection — Conformance testing methodology and framework — Part 3: The Tree and Tabular Combined Notation (TTCN)*

[5] ISO/IEC 9646-4, *Information technology — Open Systems Interconnection — Conformance testing methodology and framework — Part 4: Test realization*

[6] ISO/IEC 9646-5, *Information technology — Open Systems Interconnection — Conformance testing methodology and framework — Part 5: Requirements on test laboratories and clients for the conformance assessment process*

[7] ISO/IEC 9646-6, *Information technology — Open Systems Interconnection — Conformance testing methodology and framework — Part 6: Protocol profile test specification*

[8] ISO/IEC 9646-7, *Information technology — Open Systems Interconnection — Conformance testing methodology and framework — Part 7: Implementation Conformance Statements*

(*Continued from second cover*)

International Standard	*Corresponding Indian Standard*	*Degree of Equivalence*
ISO/IEC 14443-2 : 2020 Cards and security devices for personal identification — Contactless proximity objects — Part 2: Radio frequency power and signal interface	IS/ISO/IEC 14443-2 : 2020 Cards and security devices for personal identification contactless proximity objects: Part 2 Radio frequency power and signal interface	Identical
ISO/IEC 14443-3 : 2018 Cards and security devices for personal identification — Contactless proximity objects — Part 3: Initialization and anticollision	IS/ISO/IEC 14443-3 : 2018 Cards and security devices for personal identification — Contactless proximity objects — Part 3: Initialization and anti-collision	Identical
ISO/IEC 14443-4 : 2018 Cards and security devices for personal identification — Contactless proximity objects — Part 4: Transmission protocol	IS/ISO/IEC 14443-4 : 2018 Cards and security devices for personal identification — Contactless proximity objects — Part 4: Transmission protocol	Identical

For the purpose of deciding whether a particular requirement of this standard is complied with, the final value, observed or calculated, expressing the result of a test or analysis, shall be rounded off in accordance with IS 2 : 2022 ‘Rules for rounding off numerical values (*second revision*)’. The number of significant places retained in the rounded off value should be same as that of the specified value in this standard.

Bureau of Indian Standards

BIS is a statutory institution established under the *Bureau of Indian Standards Act*, 2016 to promote harmonious development of the activities of standardization, marking and quality certification of goods and attending to connected matters in the country.

Review of Indian Standards

Amendments are issued to standards as the need arises on the basis of comments. Standards are also reviewed periodically; a standard along with amendments is reaffirmed when such review indicates that no changes are needed; if the review indicates that changes are needed, it is taken up for revision. Users of Indian Standards should ascertain that they are in possession of the latest amendments or edition by referring to the website-www.bis.gov.in or www.standardsbis.in.

This Indian Standard has been developed from Doc No.: LITD 16 (19783).

Amendments Issued Since Publication

Amend No.	Date of Issue	Text Affected

BUREAU OF INDIAN STANDARDS

Headquarters:

Manak Bhavan, 9 Bahadur Shah Zafar Marg, New Delhi 110002
Telephones: 2323 0131, 2323 3375, 2323 9402 *Website*: www.bis.gov.in

Regional Offices:		*Telephones*
Central	: 601/A, Konnectus Tower -1, 6th Floor, DMRC Building, Bhavbhuti Marg, New Delhi 110002	2323 7617
Eastern	: 8th Floor, Plot No 7/7 & 7/8, CP Block, Sector V, Salt Lake, Kolkata, West Bengal 700091	2367 0012 2320 9474
Northern	: Plot No. 4-A, Sector 27-B, Madhya Marg, Chandigarh 160019	265 9930
Southern	: C.I.T. Campus, IV Cross Road, Taramani, Chennai 600113	2254 1442 2254 1216
Western	: 5th Floor/MTNL CETTM, Technology Street, Hiranandani Gardens, Powai Mumbai 400076	25700030 25702715

Branches : AHMEDABAD, BENGALURU, BHOPAL, BHUBANESHWAR, CHANDIGARH, CHENNAI, COIMBATORE, DEHRADUN, DELHI, FARIDABAD, GHAZIABAD, GUWAHATI, HARYANA (CHANDIGARH), HUBLI, HYDERABAD, JAIPUR, JAMMU, JAMSHEDPUR, KOCHI, KOLKATA, LUCKNOW, MADURAI, MUMBAI, NAGPUR, NOIDA, PARWANOO, PATNA, PUNE, RAIPUR, RAJKOT, SURAT, VIJAYAWADA.

Published by BIS, New Delhi