FINAL DRAFT

AMENDMENT

ISO/IEC 10373-6:2011 FDAM 5

ISO/IEC JTC **1**

Secretariat: **ANSI**

Voting begins on:
2013-10-02

Voting terminates on:
2013-12-02

Identification cards — Test methods —

Part 6: **Proximity cards**

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

Cartes d'identification — Méthodes d'essai —

Partie 6: Cartes de proximité

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

RECIPIENTS OF THIS DRAFT ARE INVITED TO SUBMIT, WITH THEIR COMMENTS, NOTIFICATION OF ANY RELEVANT PATENT RIGHTS OF WHICH THEY ARE AWARE AND TO PROVIDE SUPPORTING DOCUMENTATION.

IN ADDITION TO THEIR EVALUATION AS BEING ACCEPTABLE FOR INDUSTRIAL, TECHNOLOGICAL, COMMERCIAL AND USER PURPOSES, DRAFT INTERNATIONAL STANDARDS MAY ON OCCASION HAVE TO BE CONSIDERED IN THE LIGHT OF THEIR POTENTIAL TO BECOME STANDARDS TO WHICH REFERENCE MAY BE MADE IN NATIONAL REGULATIONS.

Reference number
ISO/IEC 10373-6:2011/FDAM 5:2013(E)

Foreword

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

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

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

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

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

Identification cards — Test methods — Part 6: Proximity cards

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

Page 22

Add new subclause

"

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

See Annex J.

"

After Annex I

Add following new annex:

Annex J
(normative)

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

J.1 Overview

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

NOTE Future revisions of ISO/IEC 14443 and ISO/IEC 10373-6 may specify new NPV tolerance and phase noise values with corresponding test methods.

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

J.2.1 PCD Tests

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

J.2.1.1 PCD phase range and waveform characteristics

J.2.1.1.1 Purpose

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

J.2.1.1.2 Test procedure

Apply the procedure defined in 7.1.4.2 with the following adaptations:

— After the activation of a bit rate of 3*fc*/4, *fc*, 3*fc*/2 or 2*fc*, the PCD shall transmit an $I(0)_0$(TEST_COMMAND1(1)).

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

J.2.1.1.3 Test report

The test report shall give the measured PR, ISI_m, ISI_d and the normalized differential phase noise values of the PCD field, within the defined operating volume in unloaded and loaded conditions.

NOTE J.3.13 gives some example test reports.

J.2.2 PICC Tests

J.2.2.1 PICC reception

J.2.2.2 Purpose

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

J.2.2.3 Test conditions

Four test conditions are defined at the border of the PICC signal parameters as defined in ISO/IEC 14443-2:2010/Amd 5. A low pass filtered pseudo-random white noise as defined in J.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 14443-2:2010/Amd 5. The test conditions are created using the test PCD assembly in combination with digital pre-conditioning of the transmitted APVs as shown in J.4:

— 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 14443-2:2010/Amd 5;

— 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 14443-2:2010/Amd 5;

— 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 14443-2:2010/Amd 5;

— 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 14443-2:2010/Amd 5.

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

NOTE 2 J.4 informatively describes how to create the above 4 conditions in the base-band domain (on the complex envelope of the signal).

These 4 test conditions shall be tested at least using H_{min} and H_{max}.

J.2.2.4 Test procedure

For each supported bit rate of 3*fc*/4, *fc*, 3*fc*/2 and 2*fc*, the PICC shall operate under the defined conditions after the selection of a bit rate. This PICC shall respond correctly to an $I(0)_0$(TEST_COMMAND1(1)) transmitted at the specified bit rate.

The activation of the bit rates uses S(PARAMETERS) mechanism as defined in ISO/IEC 14443-4:2008/Amd 3.

NOTE For a frame size higher than 256 bytes a frame with error correction as defined in ISO/IEC 14443-4:2008/Amd 4 should be used.

J.2.2.5 Test report

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

J.3 PCD waveform characteristics analysis tool for bit rates of 3*fc*/4, *fc*, 3*fc*/2 and 2*fc*

J.3.1 Overview

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

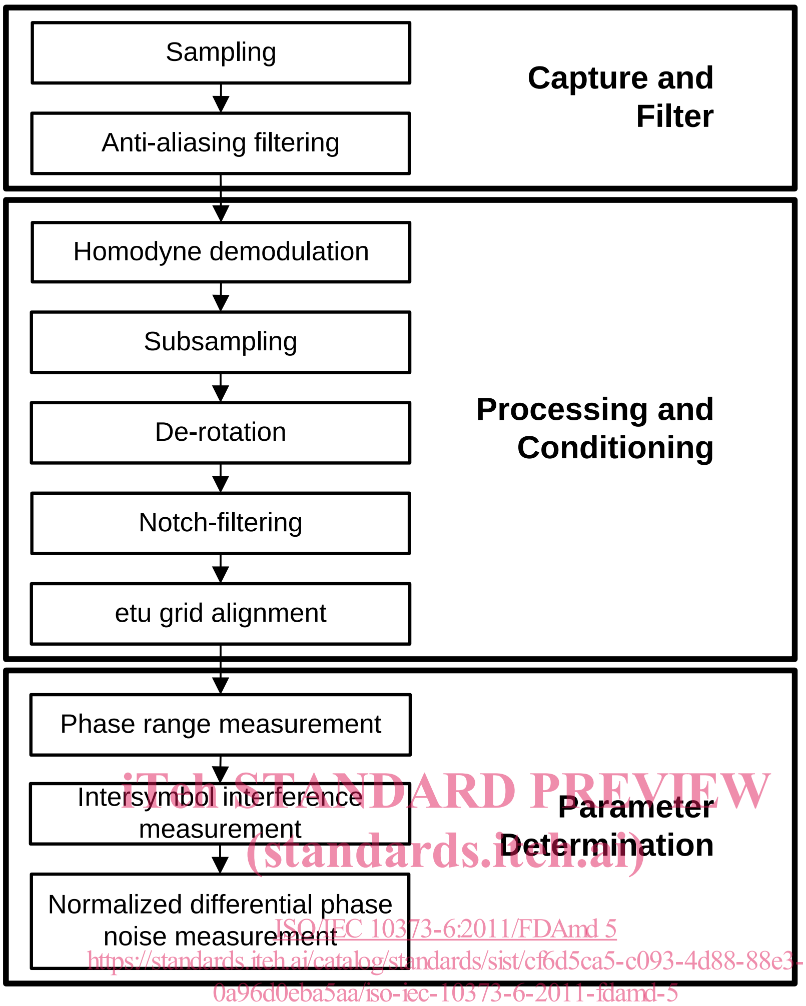

Figure J.1 — Block diagram of the analysis tool for bit rates of 3*fc*/4, *fc*, 3*fc*/2 and 2*fc*

Each block is separately described in the subsequent clauses.

J.3.2 Sampling

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

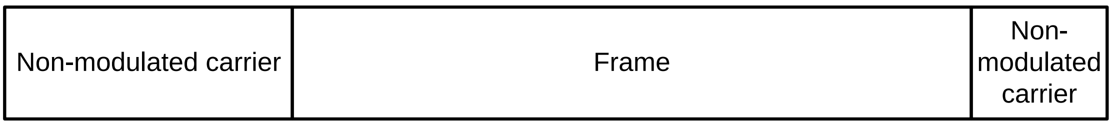

Figure J.2 — Non-modulated carrier followed by one frame, followed by non-modulated carrier

J.3.3 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 J.3.

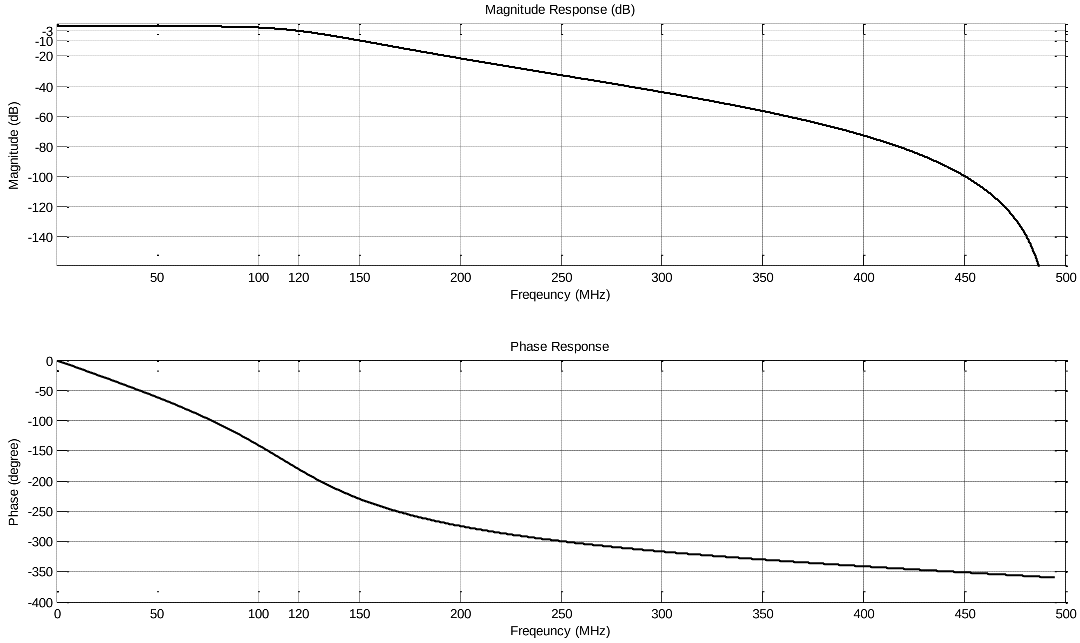

Figure J.3 — Anti-aliasing filter characteristics

J.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 J.4).

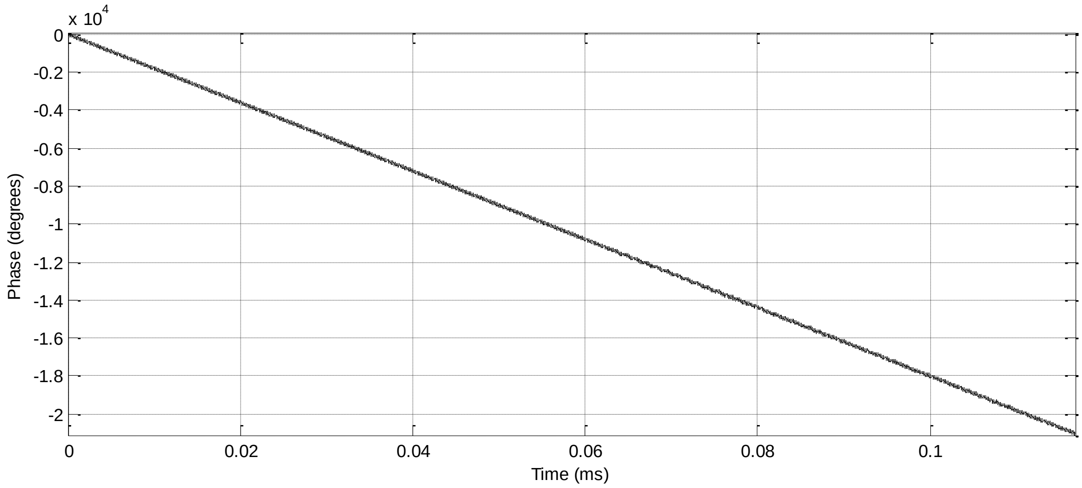

Figure J.4 — Example phase signal over time after homodyne demodulation

J.3.5 Subsampling

The phase signal shall be sub-sampled to an integer number multiple of *fc* using linear interpolation. The integer number shall be at least 32.

J.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 J.5).

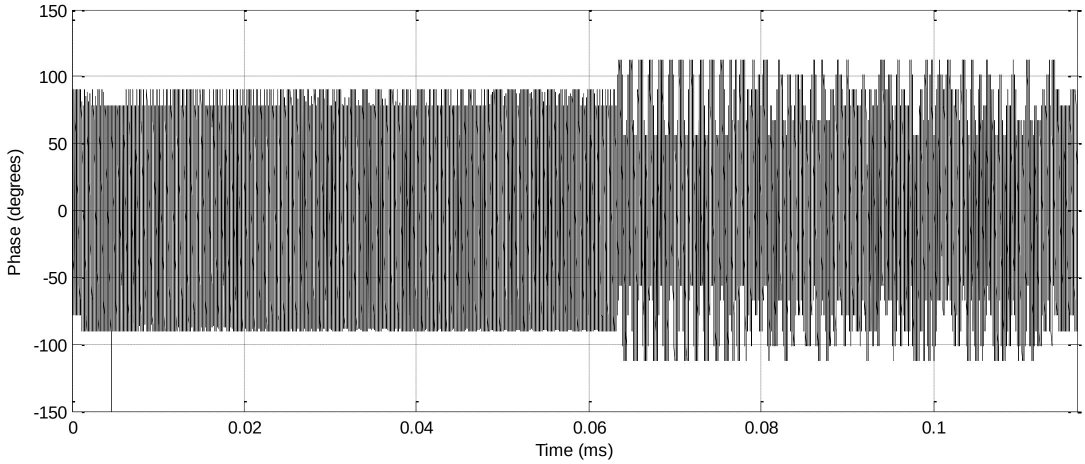

Figure J.5 — Example phase signal after de-rotation

J.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/*fc* (see Figure J.6).

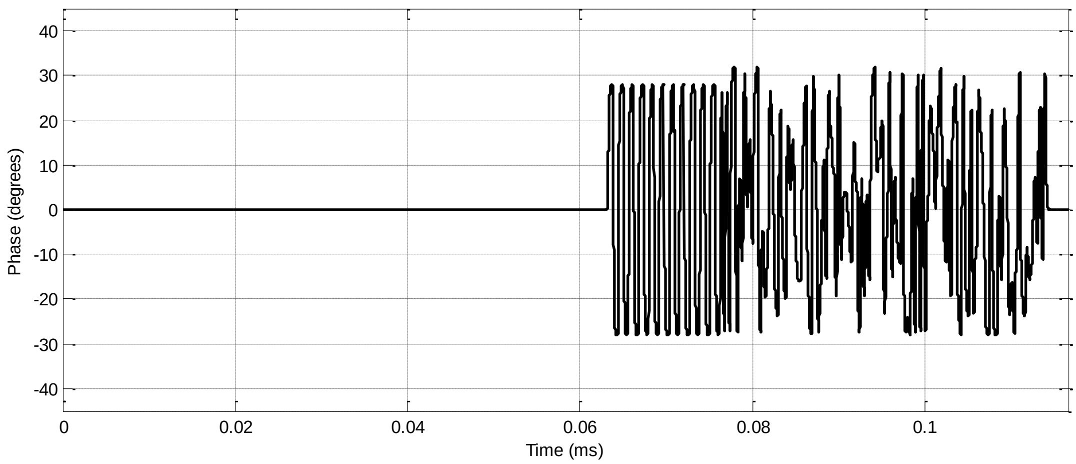

Figure J.6 — Example phase signal after filtering

J.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 14443-2:2010/Amd 5. The etu grid alignment is carried out by maximizing the computed correlation, using the phase signal and the reference phase signal.

J.3.9 Phase range measurement

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

J.3.10 Intersymbol interference measurement

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

J.3.11 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 14443-2:2010/Amd 5.

J.3.12 Program of the PCD waveform characteristics analysis tool for bit rates of 3*fc*/4, *fc*, 3*fc*/2 and 2*fc* (informative)

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

This ANSIC implementation 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 3.14159265358979323846264338327950
#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                                                        ***/
/***  DESCIRPTION:                                                       ***/
/***    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                                                         ***/
/***  DESCRIPTION: 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
```