

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

Document type: Text for DAM ballot

Title: Notification of ballot: ISO/IEC 10373-6:2011/DAM 4 — Identification cards — Test methods — Part 6: Proximity cards — AMENDMENT 4: Bit rates higher than fc/16 and up to fc/2

Status:

BACKWARD POINTER: N 4252, N 4309, N 4403 and N 4404

STATUS: Ballot to be processed by ITTF for DIS ballot. Start date to be informed by ISO.

WORK ITEM:60116

Date of document: 2011-12-06

Expected action: INFO

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

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

ISO/IEC JTC 1/SC 17/WG 8 N XXXX

Date: 2011-11-14

ISO/IEC 10373-6:2011/DAM 4

ISO/IEC JTC 1/SC 17/WG 8

Secretariat: DIN

Identification cards — Test methods — Part 6: Proximity cards

AMENDMENT 4

Bit rates of $fc/8$, $fc/4$ and $fc/2$ and frame size from 512 to 4096 bytes

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

Warning

This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an International Standard.

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.

AMENDEMENT 4

Débits binaires de $fc/8$, $fc/4$ et $fc/2$ et taille de trame de 512 à 4096 octets

Document type: International Standard
Document subtype: Amendment
Document stage: (40) Enquiry
Document language: E

Copyright notice

This ISO document is a Draft International Standard and is copyright-protected by ISO. Except as permitted under the applicable laws of the user's country, neither this ISO draft nor any extract from it may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, photocopying, recording or otherwise, without prior written permission being secured.

Requests for permission to reproduce should be addressed to either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Reproduction may be subject to royalty payments or a licensing agreement.

Violators may be prosecuted.

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 4 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*.

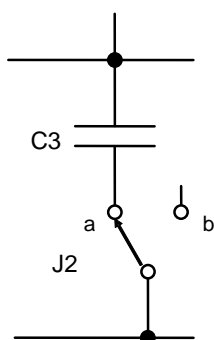
Identification cards — Test methods — Part 6: Proximity cards

Amendment 4: Bit rates of $fc/8$, $fc/4$ and $fc/2$

Page 10, Figure 5

Replace component "C3" in Figure 5 with:

"



"

Page 11, 5.4.2

Add the following two paragraphs at the end of 5.4.2:

"Position 'a' of J2 shall be used for testing bit rates up to $fc/16$.

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

Page 20, 7.2.2.3.1

Replace first paragraph with:

"Three test conditions are defined with timings at the border of the PICC modulation waveform timing parameters zone defined in ISO/IEC 14443-2, 9.1.2."

Page 20, 7.2.2.3.2

Replace second, third and fourth paragraph with:

"For each optional bit rate supported by the PICC, the PICC shall operate under the conditions defined in 7.2.2.3.1 after selection of that optional bit rate. This PICC shall respond correctly to an I-block transmitted at that optional bit rate."

Page 20

Renumber existing subclause 7.2.2.4 to 7.2.2.5 and add new subclause 7.2.2.4:

"

7.2.2.4 PICC Type A or Type B for bit rates of $fc/8$, $fc/4$ and $fc/2$

See 7.2.2.3.

"

Page 34

Add new subclause E.2.1 and move second sentence of E.2 and Figure E.2 to this subclause:

"

E.2.1 Sampling for bit rates up to $fc/16$

"

Page 35

Add new subclause E.2.2:

"

E.2.2 Sampling for bit rates of $fc/8$, $fc/4$ and $fc/2$

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.

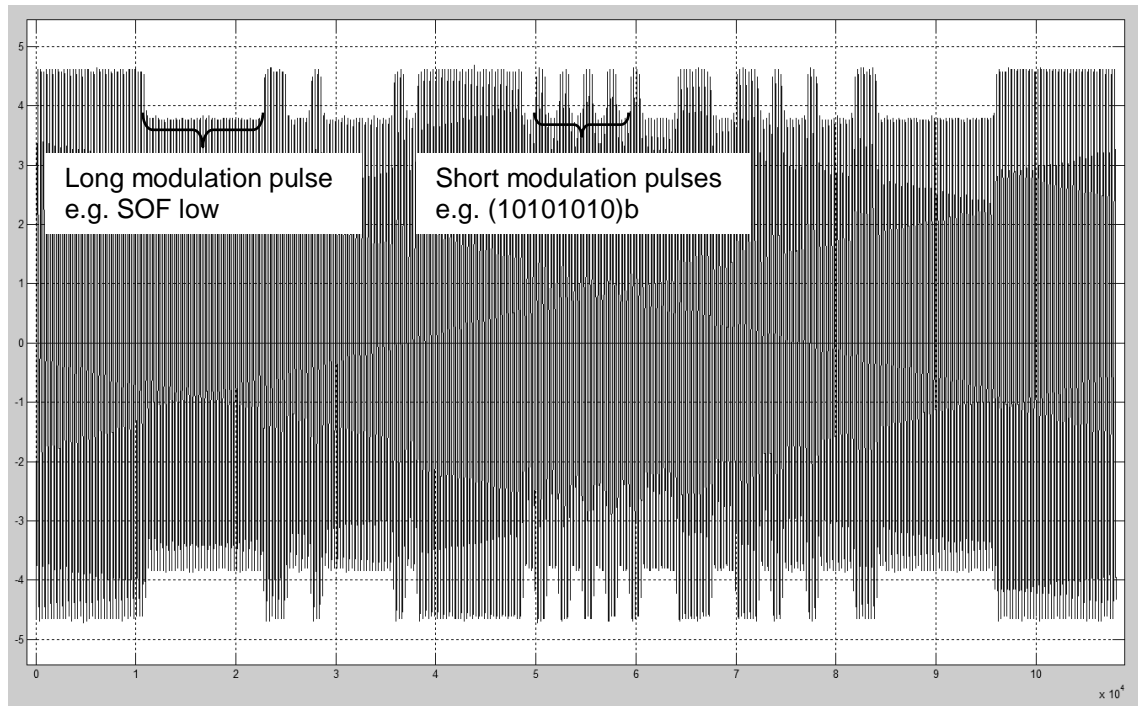


Figure E.3 — Modulation pulses

"

Page 35

Add new subclause E.3.1 and move existing paragraph of E.3 and Figure E.3 to this subclause:

"

E.3.1 Filtering for bit rates up to $fc/16$

"

Page 35

Add new subclause E.3.2:

"

E.3.2 Filtering for bit rates of $fc/8$, $fc/4$ and $fc/2$

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.

"

Page 36

Add new subclause E.5.1 and move existing paragraph of E.5 and Figure E.4 to this subclause:

"

E.5.1 Envelope smoothing for bit rates up to $fc/16$

"

Page 36

Add new subclause E.5.2:

"

E.5.2 Envelope smoothing for bit rates of $fc/8$, $fc/4$ and $fc/2$

No smoothing of signal envelope shall be applied.

"

Page 36, E.6

Add the following paragraph and figure after the paragraph:

"

For bit rates of $fc/8$, $fc/4$ and $fc/2$ and not settled envelope signals the minimum modulation index m_{min} shall be determined out of the part of the modulation sequence containing short modulation pulses as defined in Figure E.5.

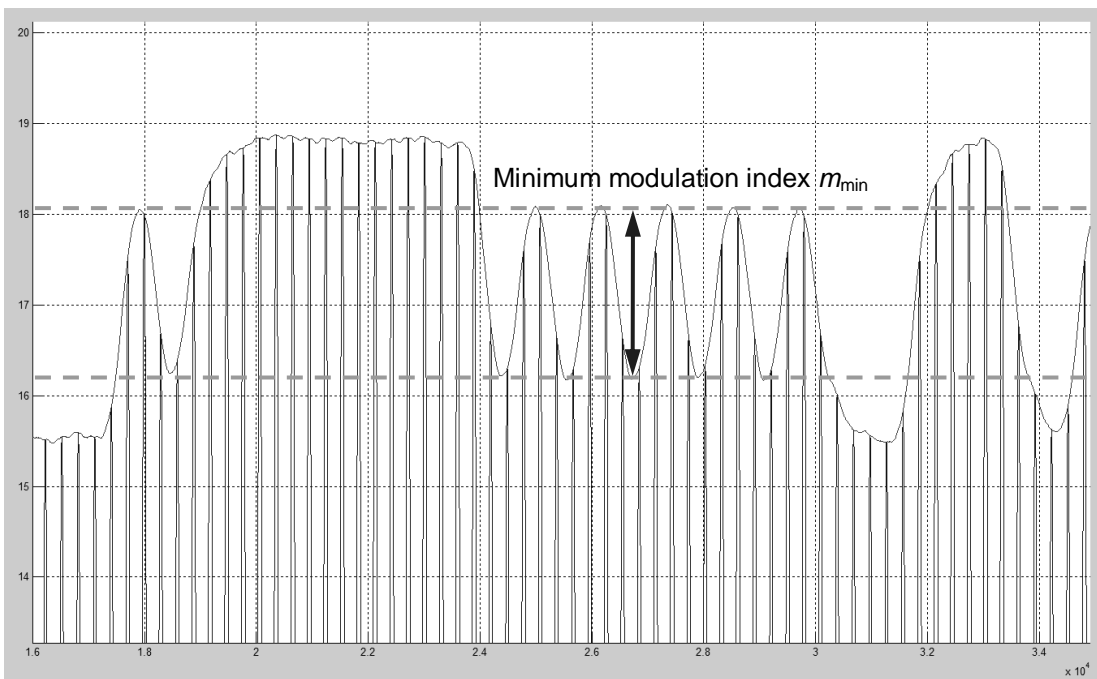


Figure E.5 — Minimum modulation index m_{min}

"

Page 36, E.7

Add at the end of paragraph:

"

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

"

Page 38

Add after line "double b; //Type B":

```
"double bVHBR; //Bit rates of fc/8, fc/4 and fc/2"
```

Page 53

Replace function "createtime" with:

```
"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 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 tploone)"
```

Add after line "new->b=b;":

```
"new->bVHBR=bVHBR;"
```

Page 59

Add following function:

```
"// Finds the value of m_min for bit rates of fc/8, fc/4 and fc/2"
```

```
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*0.2;
Hmax_cm=Hmax*0.95;
b_cm=Hmin*1.05;
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;
}
}
"

```

Page 59

Replace function "envfilt" with:

```

"int envfilt(int rate, double *output, double *toutput, int filterlength, double
tini, double tend, int lengthp, double *envelope)"

```

Page 60

Replace line "LinearConvolution(cof, output, envelope, lengthf, lengthp);" with:

```

"
    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);
    }
"

```

Page 62

Add in function "tfinder" after line " int i=0;" the following :

```
"  
  
    double *toutpu2=NULL;  
    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;  
  
    toutput2=toutput;  
  
"
```

Page 65

Replace line "createtime (timeres,0,0,0,0,0,0,0,t1,.....,0,0,0,0);" with:

```
"createtime (timeres,0,0,0,0,0,0,0,t1,t1startind,t1start,t1endind,t2,t2startind,  
t2start,t3,t3end,t3endind,t4,t4endind,0,0,0,0,0,0,0,0);"
```

Page 67

Replace line "createtime (timeres,0,0,0,0,0,0,0,.....,a,tploone) ;" with:

```
"createtime (timeres,0,0,0,0,0,0,0,t1,t1startind,t1start,t1endind,0,0,0,0,0,0,0,  
0,t5,t5startind,t6,t6end,t6endind,a,tploone) ;"
```

Page 67

Replace complete code for 'case B' with:

```
"  
    {  
        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);
        }
        else
        {
            fprintf(stdout,"Monotony not fulfilled\n");
        }

        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)
                {
                    tphone=crosses_WORK->time;      // Temporary values
are stored for future use
                    vphone=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
        {
            fprintf(stdout,"Monotony not fulfilled\n");
        }

        tf=tploone-tphone;          // Definitive values are calculated and
stored for display

```

```

tr=tphitwo-tplotwo;
tfstartind=tphone; // Other important values for the coming
functions
tfendind=tploone;
trstartind=tplotwo;
trendind=tphitwo;

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);
}
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);

overshoot
tfstartind=crosses2->time; // Reused as start point for
tfendind=crosses->time; // and undershoot

// 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;

    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;

        VHBR_step=0.0;    // Reset Counters
        VHBR_tr=0.0;
        counter=0;
        rev_counter=0;
        crosses=crosses->sig;
    }
}
}
}

```



```

        // Calculate and Save Parameters
        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);
        }
        break;
    }
}
break;
}
}
}
"

```

Page 71

Replace:

```

"while (index_samples<=samples)
{
if (env2[index_samples]>above)"

```

with:

```

"while (env2[index_samples]!=0)
{
if (env2[index_samples]>above)"

```

Page 72

Replace:

```

"while (toutput[index_samples]<(timesp->trstartind))
{
if (env2[index_samples]<above_b && env2[index_samples]!=0)"

```

with:

```

"while (env2[index_samples]!=0)
{
if (env2[index_samples]<above_b)"

```

Page 72

Replace function declaration "display" with:

```
"void display(char type, int rate, SHOOTREADER *shootreader2, TIMES *timesp,
double Hmax, double m, double mmin)"
```

Page 73

Add after line "fprintf(stdout, "---RESULTS-----\n");":

```
"if (rate<=848)
    fprintf(stdout, "Type B - bit rate %d\n", rate);
else
    fprintf(stdout, "Type VHBR - bit rate %d\n", rate);"
```

Page 74

Add after line " fprintf(stdout, "m = %f %% \n", m);":

```
"if((rate==1700 || rate==3400 || rate==6800))
    fprintf(stdout, "m_min = %f %% \n", mmin);"
```

Page 74

Replace function "main" with:

```
" 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 *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));

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 (A, B or VHBR)");
    else if ((type=='A' || type=='B') && (rate!=106 && rate!=212 && rate!=424
&& rate!=848))
        fprintf(stdout, "Wrong bit rate (106, 212, 424, 848)");
    else if ((type=='V') && (rate!=1700 && rate!=3400 && rate!=6800))
        fprintf(stdout, "Wrong bit rate (1700, 3400, 6800)");
    else
    {
        if (type=='V')
            type='B';
        pointfile=fopen(argv[3],"r");
        input_u2=fopen("pre_Hilbert.txt","w");           // modified-
intermediate amplitude vector

        if(pointfile!=NULL && 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)) // 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;

samplesp=datacheck(posval, negval, samplesp, tlast, pointfile);

tlast=tfilter[samplesp];

//2. DATA FILTER BANDWIDTH (10 MHz OR 20 MHz DEPENDING ON BIT RATE)
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_coefs(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);
        }
        for (fi=0; fi<(samplesp-7); fi++) // Reading the lines of
the voltage input file
        {
                val=voutput[fi];
                read_line (pointfile,voltstr,timestr);
                fprintf(input_u2,"%s,%f\n",timestr,val);
                length++;
        }

//3. HILBERT TRANSFORM AND THE COMPLEX ENVELOPE
        rewind(input_u2);
        hilbert("pre_Hilbert.txt"); // performs Hilbert transform

        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)) // 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);

//4. USING A SMOOTHING FILTER (MOV. AVG) TO REDUCE THE NOISE
        filterlength=3;
        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);

//7. M_min FOR BIT RATES OF fc/8, fc/4 AND fc/2
if (rate==1700 || rate==3400 || rate==6800)
    Mminfinder(envelope, Hmax, Hmin, &HmaxVHBR, timesp,
length_total);

//8. 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];

//9. 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);
    monocheck(envelope, toutput, Hmax2, timesp2, rate, type);
    // The parameters of the alternate envelope are calculated
    overshoot(timesp2, Hmax2, envelope, toutput, rate, type, samples,
shootreader2); // 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,
shootreader2); // This time the over- and undershoots are found
    }

    //10. MODULATION
    m=modulation(type, Hmax, timesp->b);
    if((type=='B') && (rate==1700 || rate==3400 || rate==6800))
        mmin=modulation(type, HmaxVHBR, timesp->bVHBR);

    //11. DISPLAY
    display(type, rate, shootreader2, timesp, Hmax, m, mmin);
}
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;
}
"

```

Page 80

Replace Annex F with:

"

Annex F (informative)

Program for the evaluation of the spectrum

"The following program written in C language gives an example for the calculation of the magnitude of the spectrum from the PICC.

```

/*****
/**** 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 Barlett 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);
        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 *****/
"

```

Page 112, G.3.7.2

Replace first paragraph with:

"Perform the following steps for each FSDI defined in ISO/IEC 14443-4:"

Page 127, G.4.7.2

Replace first paragraph with:

"Perform the following steps for each Maximum Frame Size Code defined in ISO/IEC 14443-3:"

Page 161, H.2.7

Replace last sentence of first paragraph with:

"This test shall be executed for at least FSCI set to '0', '1', '8' and 'C'."

Page 166, H.3.2

Replace second paragraph with:

"This test shall be executed for at least Maximum Frame Size Code set to '0', '1', '8' and 'C'."

Add at the end of subclause I.2.2.2:

"I.2.3 Procedure for bit rate selection using S(PARAMETERS) blocks

The following procedure shall be repeated for all values of bit rates information bytes defined in Table I.3 for PCDs using S(PARAMETERS) mechanism for bit rate selection.

- a) The UT performs the protocol activation procedure according to H.1.8.2 for Type A or H.1.8.3 for Type B.

NOTE 1 If the PCD is embedded 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.3.

NOTE 2 Only a special set of all possible bit rates is tested. Second 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.

NOTE 3 This block may also be I(1)₀, or R(NACK) in case of PICC presence check method 2a as described in ISO/IEC 14443-4:2008, 7.5.5.2.

- 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 4 The following steps may not be applicable when a PCD is embedded in a product:

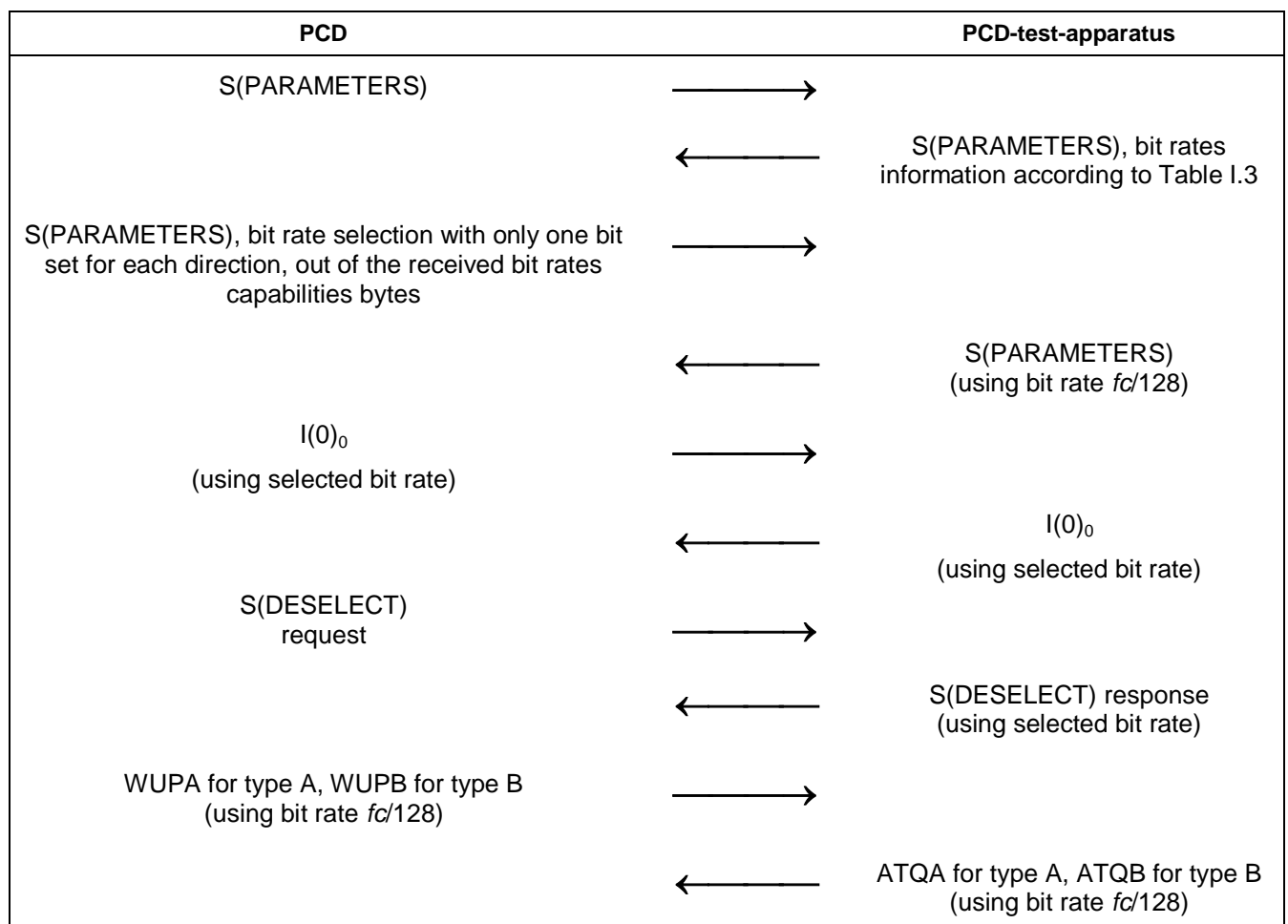
- h) The PCD shall send an S(DESELECT) request using the bit rate selected.
- i) The PCD-test-apparatus sends a valid S(DESELECT) response using the bit rate selected. Check, if the answer from the PCD-test apparatus is accepted by the PCD.
- j) The PCD shall send a valid WUPA for Type A or WUPB for Type B using the bit rate $f_c/128$.
- k) The PCD-test-apparatus answers with a valid ATQA for Type A or ATQB for type B.

Table I.3 — Supported bit rates

First byte PCD to PICC								First 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	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	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
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

First byte PCD to PICC								First byte PICC to PCD							
b8	b7	b6	b5	b4	b3	b2	b1	b8	b7	b6	b5	b4	b3	b2	b1
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

Scenario I.3 — Bit rate selection using S(PARAMETERS) blocks, Procedure 3



I.2.3.1 Expected result

The PCD shall behave as described in Scenario I.3 in each of the 34 test cases.

I.2.3.2 Test report

If the PCD behaves valid according to Scenario I.3 in each of the 34 test cases, then this test passed. The test report should document the bit rates chosen by the PCD in each of the 34 test cases."