ISO/IEC JTC1/SC17/WG8 N 1856
ISO/IEC JTC1/SC17 N 4403

Disposition of comments on:
CD ISO/IEC 10373-6:2011/PDAM4 — Identification cards — Test methods — Part 6: Proximity cards — AMENDMENT 4: Bit rates higher than fc/16 and up to fc/2

Reference documents:

Ballot is in SC17 N 4252 = WG8 N 1827
Ballot Result is in SC17 N 4309 = WG8 N 1841

Project Editor:
Peter Raggam, Germany

The following pages provide the details of the comments and detailed information about their resolutions, how WG8 had tried to resolve each received comment from the CD Ballot (PDAM) at the WG8 meeting held in Song-Do, Korea, on 2011-09-28/30.

According to the advice from the SC17 Secretariat WG8 decided by WG8 Resolution 50.09 (contained in WG8 N 1847) to issue the new text of 10373-6:2011/Amd.4, i.e. WG8 N 1855, for FCD 10373-6:2011/FPDAM4 balloting.

1	2	3	4	5	6	7
MB[1]	**Clause / Subclause/ Annex/Figure/Table** (e.g. 3.1, Table 2)	**Paragraph/ List item/ Note/** (e.g. Note 2)	**Type of com-ment**[2]	**Comment (justification for change)**	**Proposed change**	**Secretariat observations** on each comment submitted
This file contains all comments to ISO/IEC 10373-6 PDAM 4 **Comments are ordered along clauses.**						
NL1	Whole		GE	The contents of the CD ballots contained specifications for ASK and PSK for each individual very high data rate. There is substantial need for the PSK technology in the Netherlands which is now discriminated by WG8 during the comment resolution. The provided specification for PSK on fc/8, fc/4 and fc/2 were eliminated against the explicit request of NL to keep them in the standard. This decision may lead to the situation that a leading VHBR technology for PICCs will be deployed outside of ISO 14443 specifications. The Dutch industry which may serve the vast majority of the relevant world market requests PSK technology being treated in the same and fair way as the alternative.	Turn back and specify both ASK and PSK methods for all data rates (e.g. fc/8, fc/4, fc/2, 3fc/4 and fc)	Rejected. See NL1 of DoC 14443-2 Amd3
NL2	Whole		GE	The separation of ASK and PSK specifications into 8 separate documents is rather confusing than supportive for efficient reading and implementation. Also it was never authorized by ISO and in the related NP approvals	Please specify both ASK and PSK methods for all data rates (e.g. fc/8, fc/4, fc/2, 3fc/4 and fc) in the same amendments, one for each part of the standard.	Rejected. See NL1 of DoC 14443-2 Amd3
UK2	General		ed	As this amendment introduces is a new technique (akin to the Type A Type B situation several years ago) it should be clearly recognisable in the standard as significantly different from existing techniques. The hope is that this will avoid the very real	The substance of the proposed changes for this new technique should be contained in a separate ANNEX (Type X) within each part of the specification. Where changes are inserted in the main body of the standard these should	Acknowledged

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China) ** = ISO/CS editing unit

2 **Type of comment**: ge = general te = technical ed = editorial

NB Columns 1, 2, 4, 5 are compulsory.

				possibility of confusion and possible fragmentation of the market place.	reference the Type X ANNEX.	
JP1	Whole documents		ge	It is not clear at PDAM ballot stage whether there are any known patents regarding ISO/IEC 10373-6/Amd.4. Even though ISO/IEC takes no position concerning the evidence, validity and scope of the patents, the patents known at this time should be disclosed.	Disclose the patent list regarding ISO/IEC 10373-6/Amd.4.	Will be resolved at SC17 plenary Songdo
JP2	Page ii , 1	Titles	ed	Titles are different from the basic document.	Replace “Bit rate higher than *fc*/16 and up to *fc*” by “Bit rate of *fc*/8, *fc*/4 and *fc*/2”	Accepted
FR1	Title	Front page and 1st page	ED	Consistency with base standard amendments on the same bit rates	Replace "Bit rates higher than fc/16 and up to fc/2" with "Bit rates of fc/8, fc/4 and fc/2" Replace "Débits binaires supérieurs à fc/16 jusqu'à fc/2" with "Débits binaires de fc/8, fc/4 et fc/2"	Accepted
UK1	Page ii	Copyright notice	ed	ISO copyright notice reminder to be filled in	Also Note to ISO. Correct spelling in ISO template Replace manger with manager	Accepted
FR2	all		ed	The amended document (ISO/IEC 10373-6:2011) should not be mentioned	Replace "*Page XX of ISO/IEC 10373-6:2011,* " with "*Page XX,* " in the whole amendment	Accepted
JP3	5.4.2	Table 2	te	For future testing all the bit rates with one Reference PICC, it is more convenient to select “with C3(Type-A and Type-B)“ or “without C3(bit rates higher than fc/16 and up to fc/2)” by using jumper switch	Replace “C3” in Figure 5 by the diagram described in [JP Annex-1], instead of table note for component C3.	Resolved: Adding figure where C3 is selectable depending on bitrates.
FR3	Table 2		ED	The suppression of capacitor may be useful for any bit rate higher than fc/16	Delete "and up to fc/2" (If needed then replace "higher than fc/16 and up to fc/2" with "of fc/8, fc/4 and fc/2")	Resolved: Suppression only for bit rates of fc/8, fc/4 and fc/2 relevant PICC to PCD direction. See JP3
FR4			TE	There is no PICC reception test	Add a PICC reception test similar to 7.2.2.3: 7.2.2.4 PICC Type A or Type B, bit rates of fc/8, fc/'4 and fc/2 See 7.2.2.3	Resolved
FR5	E.2.1		ED	The following instruction is complex and not consistent with equivalent simpler instruction in 14443-2/PDAM5, 9.1.3 *Page 34 of ISO/IEC 10373-6:2011* Add new sub clause E.2.1 and move existing paragraph of E.2 and Figure E.2 to this sub clause:	Replace this instruction with: *Page 34, E.2* Add a subclause title: " E.2.1 Sampling for bit rates up to fc/16	Resolved

				" E.2.1 Sampling for bit rates up to fc/16 " Besides "subclause" is one word with no space (to be changed in the whole document)	"	
FR6	E.2.2	Title	ED	No need to state "up to fc/2"	Delete "up to fc/2" (If needed then replace "higher than fc/16 and up to fc/2" with "of fc/8, fc/4 and fc/2") Same comment for E.3.2 title, E.5.2 title, E.6 addition and E.7 addition.	Accepted
FR7	E.2.2	1st paragraph	ED	The sentence is not very clear	Replace "The time and voltage data of more than one modulation pulse, preferably a complete S(DESELECT) command with at least 20 carrier periods before the first and after the last modulation pulse containing short and long modulation pulses (see Figure E.3) shall be transferred to a suitable computer." with "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."	Accepted
FR8	E.6		ed	consistency	Replace "…containing short modulation pulses (see Figure E.6)." with "…containing short modulation pulses as defined in Figure E.6."	Accepted
FR9	Annex F	Line 11 of program	Ed	Fcm/8, Fcm/4 or Fcm/2 use is not very clear	Replace "Fcm/16 for bit rates up to fc/16, Fcm/8, Fcm/4 or Fcm/2 for bit rates up to fc/2" with "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"	Accepted
FR10	Annex F	Line 22 of program	ed	English language	Replace "equidistant time" with "equidistant in time"	Accepted
FR11	Annex F	Lines 23 of program	ED	The requirements should not be in the program	Delete "minimum sampling rate: 100 MSamples/second"	Accepted
FR1	Annex G	G.3.7	TE	There are new frame sizes	Update the limits to 'C'	Resolved:

2		G.4.7 H.2.7 H.3.2			Replace this sentence "Perform the following steps for each FSDI = 0 to 8" by " Perform the following steps for each FSDI defined in ISO/IEC 14443…"	Change title and text to include higher frame sizes. H.2.7 and H.3.2 extended to include 'C'

DE 1	E.9.1. of ISO/IEC 10373-6:2011	Page 38	te	Program of the modulation index and waveform analysis tool (informative) has to be extended to support bit rates fc/8, fc/4 and fc/2.	-Add line:
					`double bVHBR; //Type B VHBR`
					after:
					`double b; //Type B`
						Accepted

DE 2	E.9.6. of ISO/IEC 10373-6:2011	Page 53	te	Program of the modulation index and waveform analysis tool (informative) has to be extended to support bit rates fc/8, fc/4 and fc/2.	Function "createtime":
					-Add variable "bVHBR" in declaration:
					`void createtime(TIMES *new, double tr, double tf, double b, double bVHBR,…)`
					- Add line:
					`new->bVHBR=bVHBR;`
					after:
					`new->b=b;`
						Accepted

DE 3	E.9.6. of ISO/IEC 10373-6:2011	Page 59	te	Program of the modulation index and waveform analysis tool (informative) has to be extended to support bit rates fc/8, fc/4 and fc/2.	-Add function "Mminfinder" (completely new function) => See Annex A	Accepted

DE 4	E.9.6. of ISO/IEC 10373-6:2011	Page 60	te	Program of the modulation index and waveform analysis tool (informative) has to be extended to support bit rates fc/8, fc/4 and fc/2.	Function "envfilt"
					-Add variable "rate" in declaration:
					`int envfilt(int rate, double *output,…`
					-Replace line:
					`LinearConvolution(cof, output, envelope, lengthf, lengthp);`
					For:
					Code in Annex B
						Accepted

DE 5	E.9.6. of ISO/IEC 10373-6:2011	Page 62	te	Program of the modulation index and waveform analysis tool (informative) has to be extended to support bit rates fc/8, fc/4 and fc/2.	New variables in function "tfinder":
					`// New variables for VHBR`
					`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`
						Accepted

					tr_Acceptedeptedum=0.0;
					double
					tf_Acceptedeptedum=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;
					insert after line:
					int i=0;	

DE 6	E.9.6. of ISO/IEC 10373-6:2011	Page 62	te	Program of the modulation index and waveform analysis tool (informative) has to be extended to support bit rates fc/8, fc/4 and fc/2.	Function „tfinder“
					-Insert line:
					toutput2=toutput;
					after
					envc2=envc;
						Accepted

DE 7	E.9.6. of ISO/IEC 10373-6:2011	Page 62	te	Program of the modulation index and waveform analysis tool (informative) has to be extended to support bit rates fc/8, fc/4 and fc/2.	Function “tfinder”:
					Replace line:
					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);
					With line:
					createtime(timeres,0,0,0,0,0,0,0,0,t1,t1startind,t1start,t1endind,t2,t2startind,t2start,t3,t3end,t3endind,t4,t4endind,0,0,0,0,0,0,0);
						Accepted

DE 8	E.9.6. of ISO/IEC 10373-6:2011	Page 67	te	Program of the modulation index and waveform analysis tool (informative) has to be extended to support bit rates fc/8, fc/4 and fc/2.	Function “tfinder”: Replace line: 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,tploon	Accepted

					e); With line: createtime(timeres,0,0,0,0,0,0,0,0,t1,t1startind,t1start,t1endind,0,0,0,0,0,0,0,0,t5,t5startind,t6,t6end,t6endind,a,tplo one);	
DE 9	E.9.6. of ISO/IEC 10373-6:2011	Page 67	te	Program of the modulation index and waveform analysis tool (informative) has to be extended to support bit rates fc/8, fc/4 and fc/2.	Replace the whole code between line: case 'B': and the end of the function with the code in Annex C	Accepted

DE 10	E.9.6. of ISO/IEC 10373-6:2011	Page 71	te	Program of the modulation index and waveform analysis tool (informative) has to be extended to support bit rates fc/8, fc/4 and fc/2.	Function “overshoot”
					-Replace lines (after case ‘B’, page 71):
					while (index_samples<=samples)
					{
					if (env2[index_samples]>above)
					with lines:
					while (env2[index_samples]!=0)
					{
					if (env2[index_samples]>above)
						Accepted

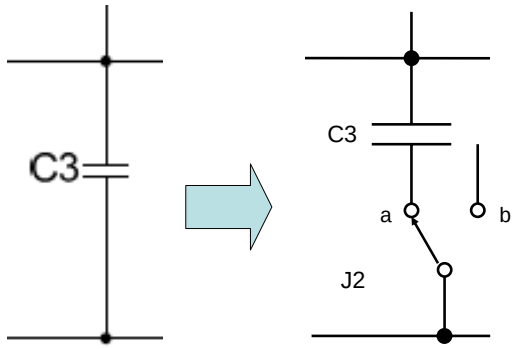
DE 11	E.9.6. of ISO/IEC 10373-6:2011	Page 72	te	Program of the modulation index and waveform analysis tool (informative) has to be extended to support bit rates fc/8, fc/4 and fc/2.	Function “overshoot”
					-Replace the following lines:
					while (toutput[index_samples]<(timesp->trstartind))
					{
					if (env2[index_samples]<above_b && env2[index_samples]!=0)
					With these other lines:
					while (env2[index_samples]!=0)
					{
					if (env2[index_samples]<above_b)
						Accepted

DE 12	E.9.6. of ISO/IEC 10373-6:2011	Page 72	te	Program of the modulation index and waveform analysis tool (informative) has to be extended to support bit rates fc/8, fc/4 and fc/2.	Function “display”: Add variable “mmin” in declaration: void display(char type, int rate, SHOOTREADER *shootreader2, TIMES *timesp, double Hmax, double m, double mmin)	Accepted

DE 13	E.9.6. of ISO/IEC 10373-6:2011	Page 74	te	Program of the modulation index and waveform analysis tool (informative) has to be extended to support bit rates fc/8, fc/4 and fc/2.	Function “display”:
					-Add lines
					`if (rate<=848)`
					`  fprintf(stdout,"Type B - Bitrate %d\n", rate);`
					`else`
					`  fprintf(stdout,"Type VHBR - Bitrate %d\n", rate);`
					after:
					`case 'B':`
					`{`
					`  fprintf(stdout,"---RESULTS-----------\n");`
					
					-Add lines
					`if((rate==1700 || rate==3400 || rate==6800))`
					`  fprintf(stdout,"m_min = %f %% \n", mmin);`
					after:
					`fprintf(stdout,"m = %f %% \n", m);`
						Accepted

DE 14	E.9.6. of ISO/IEC 10373-6:2011	Pages 74 to 79	te	Program of the modulation index and waveform analysis tool (informative) has to be extended to support bit rates fc/8, fc/4 and fc/2.	Replace Main function by new one listed in Annex D.	Accepted
IFX 1	Annex I of ISO/IEC 10373-6		Ge, te	PCD test procedure for bit rate selection using S(PARAMETERS) blocks are missing	Introduce test procedure as defined in Annex E	Resolved. Update of tables because of changes in 14443-4 Amd2

[JP Annex-1]



a: test for Type-A and Type-B
b: test for bit rates higher than fc/16 and up to fc/2

ISO/IEC 10373-6:2011, Figure5

Proposal Change

ANNEX A: New function „Mminfinder“

```
// 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*0.2;
	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;
        }
}
```

ANNEX B: Modifications in „envfilt“

```
        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);
        }
```

ANNEX C: New “tfinder” function for type B:

```
                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);
        }
        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)
                        {
                                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
        {
```

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

                        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,tfendind,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
                        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_Acceptedum=tf_Acceptedum+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_Acceptedum=tr_Acceptedum+VHBR_tr;

                                        VHBR_step=0.0;              // Reset Counters
                                        VHBR_tr=0.0;
                                        counter=0;
                                        rev_counter=0;
                                        crosses=crosses->sig;
                                 }
                          }
                   }
                   // Calculate ans Save Params.
                   tf=tf_Acceptedum/tf_counter;       // Definitive values are calculated and stored for
display
                   tr=tr_Acceptedum/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);
                   }
                   break;
            }
```

```
                }
                break;
        }
}
```

ANNEX D: New „Main“ Function

```
// 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 *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 Bitrate (106, 212, 424, 848)");
		else if ((type=='V') && (rate!=1700 && rate!=3400 && rate!=6800))
			fprintf(stdout, "Wrong Bitrate (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))                                                // 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;

                                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);
}
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))	// 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);

//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);

//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);
                                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
                        }

                        //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);
                }
                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 E:

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

The following procedure shall be repeated for all defined values in Table 6 and Table 7 in ISO/IEC 14443-4:2008, Amd.2 for PCDs using S(PARAMETERS) mechanism for bit rate negotiation.

a) The UT performs the protocol activation procedure Acceptedording to H.1.8.2 for Type A or H.1.8.3 **Error! Reference source not found.**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 given by the PCD manufacturer.

b) The PCD shall send a 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 Acceptedording to Table I.3.

NOTE 2 Only a special set of all 2^{18} possible bit rates is tested.

d) The PCD shall send a 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 Acceptedepted 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 Acceptedepted by the PCD.

j) The PCD shall send a valid WUPA for Type A or WUPB for Type B using the bit rate fc/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

PCD to PICC																PICC to PCD							
b16	**b15**	**b14**	**b13**	**b12**	**b11**	**b10**	**b9**	**b8**	**b7**	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**	**b8**	**b7**	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	1
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1	0	0	1
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	0	0	0	1
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1

PCD to PICC																PICC to PCD							
b16	**b15**	**b14**	**b13**	**b12**	**b11**	**b10**	**b9**	**b8**	**b7**	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**	**b8**	**b7**	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1
0	0	0	0	0	0	0	0	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

PCD		PCD-test-apparatus
S(PARAMETERS) ('A0 02 8A 00' CRC)	→	
	←	S(PARAMETERS), bit rates information Acceptedording to Table.3
S(PARAMETERS), bit rate selection, only one bit out of the received bit rates capabilities bytes shall be set.	→	
	←	S(PARAMETERS) (using bit rate *fc*/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 for type A, WUPB for type B (using bit rate *fc*/128)	→	
	←	ATQA for type A, ATQB for type B (using bit rate *fc*/128)

I.2.3.1 Expected result

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

I.2.3.2 Test report

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