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

Document type: Text for FDIS ballot

Title: ISO/IEC 10373-6:2011/FDAM 2 — Identification cards — Test methods — Part 6: Proximity cards — AMENDMENT 2: Test methods for electromagnetic disturbances

Status:

BACKWARD POINTER: N 3858, N 3915, N 3935, N 3936, N 4045 and N 4303.

STATUS: Notification of FDIS ballot. Ballot date to be advised by ISO.

WORK ITEM: 55977

Date of document: 2011-09-14

Expected action: INFO

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

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

WG8 N

ISO/IEC JTC 1/SC 17 xxxx

Date: 2011-05-17

ISO/IEC 10373-6:2010/FDAM 2(E)

ISO/IEC JTC 1/SC 17/WG 8

Secretariat: DIN

Identification cards — Test methods — Part 6: Proximity cards

AMENDMENT 2: Test methods for electromagnetic disturbance

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

AMENDMENT 2: Méthodes d'essai pour perturbations électromagnétiques

Document type: International Standard
Document subtype: Amendment
Document stage: (60) Publication
Document language: E

C:\Users\fdl.peters\Documents\a_bdr\a_wg8\SongDo 2011 prepa\10373-6 amd9 jetzt Amd2\FDAM10373-6_AMD2_Nxxxx_noMarks_selfNumbered.doc STD Version 2.1c2

Foreword

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

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

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

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

Amendment 2 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 2: Test methods for electromagnetic disturbance

Page 3, 3.2

Add the following symbols to the list of abbreviations and symbols:

t_{START} Start of PICC transmission

Page 12, 5.4.3

Insert the following new sub clause 5.5 after sub clause 5.4.3 and renumber all subsequent subclauses:

5.5 EMD Test Setup

5.5.1 General description

The EMD test setup contains:

- a signal generator with low phase noise, which is used to synthesize both an EMD test pattern and PCD test commands sent to the PICC under test
- the Test PCD assembly
- a signal amplitude analyzing device:
  - either a signal acquiring device (e.g. oscilloscope) and appropriate computation software
  - or a spectrum analyzer (see additional constraints in 5.5.2).

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

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

5.5.2 Computation of power versus time

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

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

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

In case of using a spectrum analyzer, the analyzer shall have at least an equivalent analysis bandwidth. It shall pass the noise floor precondition test, as defined in 5.5.3, and there shall be some additional margin of $10/fc$ on $t_{E, PICC}$ requirement and no spikes above the EMD limit.

5.5.3 Noise floor precondition test

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

The noise floor test is passed if the noise standard deviation is at least three times smaller than the EMD limit $V_{E,PICC}$, when measured as described in 5.5.3.1.

The noise standard deviation is determined by calculating the root-mean-square value of the results of the Fourier transformation, as described in 5.5.2.

NOTE This noise floor can be obtained either with a 14-bit digitizer at a sampling rate of 100 million samples per second or with an 8-bit digital oscilloscope at sampling rate of 1000 million samples per second.

5.5.3.1 Test Procedure

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

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

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

c) Place the Reference PICC into the DUT position on the Test PCD assembly, set jumper J1 to position 'b' and adjust R2 to obtain a voltage of 6 V (DC) at CON3. Alternatively, jumper J1 may be set to position 'c' and the applied voltage on CON2 is adjusted to obtain a voltage of 6 V (DC) at CON3. In both cases the operating field condition shall be verified by monitoring the voltage in the calibration coil and adjusted if necessary.

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

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

5.5.3.2 Test Report

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

Page 18, 7.1.5

Insert the following new sub clauses 7.1.6 and 7.1.7 after sub clause 7.1.5:

7.1.6 PCD EMD Immunity Test

7.1.6.1 Purpose

The purpose of this test is to determine whether the PCD is insensitive to any load modulation amplitude below $V_{E,PCD}$.

7.1.6.2 Test Procedure

a) Tune the Reference PICC to 13,56 MHz as described in 5.4.3 and switch the jumper J1 to position 'c'.

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

c) Apply and adjust a DC voltage at CON2 to obtain a DC voltage at connector CON3 of 3 V or optionally 6 V when supporting "Class 1" at that position.

d) Send the test pattern as shown in Figure Amd.2.1. The test pattern is a valid standard frame including one single byte (01011101)b. The initial load modulation amplitude V_{EMD} of the test pattern shall be sufficiently low so that the PCD detects the PICC answer sent in step e).

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

f) Increase V_{EMD} by adjusting the voltage at CON1 until the PCD does no longer detect the answer correctly. This may be determined by monitoring the next PCD command following the PICC answer, see Figure Amd.2.1.

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

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

i) Derive the current value of V_{EMD} on the Reference PICC by applying the power versus time measurement as described in 5.5.2.

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

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

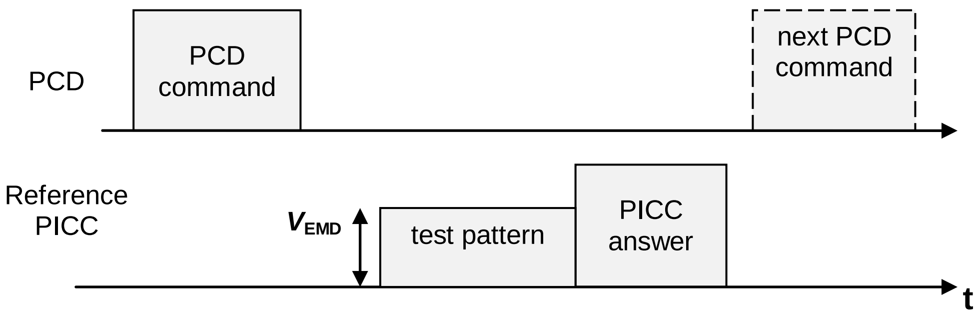

Figure Amd.2.1 — PCD immunity test (common for Type A and Type B)

7.1.6.3 Test report

The test report shall state whether the PCD was insensitive to any load modulation amplitude below $V_{E,PCD}$.

7.1.7 PCD EMD Recovery Test

7.1.7.1 Purpose

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

7.1.7.2 Test procedure

a) Tune the Reference PICC to 13,56 MHz as described in 5.4.3.

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

c) Place the Reference PICC into the DUT position on the Test PCD assembly. Switch the jumper J1 to position 'c' and adjust the DC voltage at CON2 to obtain a voltage of 6 V (DC) at CON3. The operating field condition shall be verified by monitoring the voltage on the calibration coil and the voltage adjusted if necessary.

d) Find the appropriate driving voltage at CON1 to produce a load modulation amplitude V_{LMA}, measured as defined in 7.2.1, higher than the limit for H_{min}, defined in ISO/IEC 14443-2.

e) Place the Reference PICC at a position within the operating volume of the PCD where 6 V (DC) is obtained at CON3.

f) Send in sequence, as illustrated in Figure Amd.2.2 using the $t_{E,PCD}$ associated with minimum FDT/TR0.

NOTE 1 The low EMD time $t_{E,PCD}$ is a function of FDT/TR0 as defined in of ISO/IEC 14443-3/Amd.4.

— a test pattern, which starts sending the two data bits b1 = (0)b followed by b2 = (1)b in a valid way to the PCD, but interrupts immediately after the second bit sent, as illustrated in Figure Amd.2.3 for Type A and Amd.2.4 for Type B,

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

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

— the appropriate answer to the PCD command.

g) Check if the PCD behaves in the same way as if there was no test pattern. This may be determined by monitoring the next PCD command following the PICC answer, see Figure Amd.2.2.

h) Repeat step f) and g) 10 times.

i) Repeat step f) to h) replacing minimum FDT/TR0 with maximum FDT/TR0.

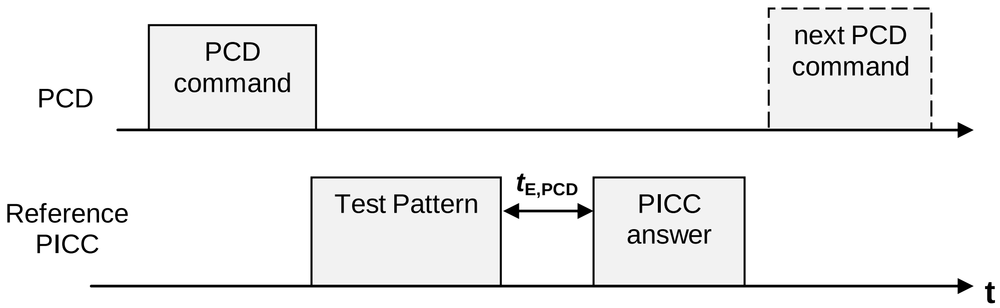

Figure Amd.2.2 — EMD recovery test sequence (common for Type A and Type B)

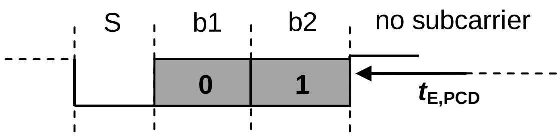

Figure Amd.2.3 — Test pattern for the EMD recovery test (Type A)

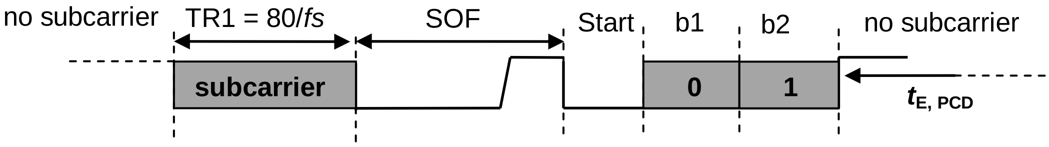

Figure Amd.2.4 — Test pattern for the EMD recovery test (Type B)

7.1.7.3 Test Report

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

Page 19, 7.2.1

Insert the following new sub clause 7.2.2 after sub clause 7.2.1 and renumber all subsequent sub clauses:

7.2.2 PICC EMD level and low EMD time test

7.2.2.1 Purpose

The purpose of this test is to determine that the PICC does not generate an electromagnetic disturbance amplitude V_{EMD} higher than $V_{E,PICC}$ during $t_{E,PICC}$ with exceptions defined in ISO/IEC 14443-2/Amd.3.

NOTE 1 The low EMD time $t_{E,PICC}$ is a function of FDT/TR0 as defined in ISO/IEC 14443-3/Amd.4.

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

7.2.2.2 Noise requirements

In order to ensure a high dynamic range and sufficient sensitivity to EMD, the noise floor precondition test defined in 5.5.3 shall be performed before this test.

7.2.2.3 Test commands

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

7.2.2.4 Test procedure

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

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

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

c) Reset the PICC by switching the RF field off and on; then if necessary send a transition of sequence commands to put the PICC in the Test Initial State, (see Annex G.3.3.2.1 for PICC Type A and Annex G.4.4.1.1 for PICC Type B).

d) Send the command to be tested.

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

f) Determine the value of $t_{E,PICC}$ from the acquired signal: if the PCD modulation is present on the trace then measure the time between the last rising edge of PCD modulation and the start of PICC subcarrier generation and calculate $t_{E,PICC}$ with the formula given in ISO/IEC 14443-3/Amd.4; if the PCD modulation is not present on the trace then $t_{E,PICC}$ equals its maximum value defined in ISO/IEC 14443-3/Amd.4.

g) Compute the signal power at the frequencies $fc + fs$ and $fc - fs$ as a function of time as defined in 5.5.2.

h) Using data obtained in step g), determine the time t_{START} corresponding to half the upper side band amplitude during the rising edge of PICC transmission. Check if the signal amplitude during the time period [$t_{START} - t_{E,PICC}$; $t_{START} - 10/fc$] complies with the requirements defined in ISO/IEC 14443-2/Amd.3.

i) Repeat step h) for the lower side band frequency.

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

7.2.2.5 Test report

The test report shall state whether the PICC EMD level during $t_{E,PICC}$ complies with the requirements defined in ISO/IEC 14443-2/Amd.3.

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

Page 194, ANNEX J, Add new ANNEX J:

Annex J
(informative)

Program for EMD level measurements

The following code in C language may be used to perform the EMD level measurements.

NOTE The output of (time, USB, LSB) may depend on compiler options and the used operating system architecture.

```
/**********************************************************************************************/
/***  This program calculates the upper side band (USB) and                                ***/
/***  lower side band (LSB) load modulation amplitudes                                     ***/
/***  versus time of a PICC for the evaluation of EMD levels                               ***/
/***  according to ISO/IEC 10373-6/Amd.2                                                   ***/
/**********************************************************************************************/
/***  Input:                                                                               ***/
/***  File in CSV format containing a table of two                                         ***/
/***  columns (time and sense coils' voltage)                                              ***/
/***  data format of input-file:                                                           ***/
/***  - one data-point per line:                                                           ***/
/***    (time[seconds], sense-coil-voltage[volts])                                         ***/
/***  - contents in ASCII, no headers                                                      ***/
/***  - data-points shall be equidistant time                                              ***/
/***  - minimum sampling rate: 100 MSamples/second                                         ***/
/***  - At least 200 microsecond before start of PICC                                      ***/
/***    sub-carrier generation                                                             ***/
/***  - At least 50 microsecond after start of PICC                                        ***/
/***    sub-carrier generation                                                             ***/
/***                                                                                       ***/
/***  example for spreadsheet file (start in next line):                                   ***/
/***   (time)      (voltage)                                                               ***/
/***  3.00000e-06,1.00                                                                     ***/
/***  3.00200e-06,1.01                                                                     ***/
/***  .....                                                                                ***/
/***                                                                                       ***/
/***  Ouput:                                                                               ***/
/***  File in CSV format containing the results                                            ***/
/***  in a table of three columns (time, USB, LSB)                                         ***/
/***                                                                                       ***/
/***                                                                                       ***/
/**********************************************************************************************/
/***  RUN:                                                                                 ***/
/***  "exefilename" filename[.csv]                                                         ***/
/**********************************************************************************************/
/***  ISO/IEC 10373-6 EMD levels Calculation                                               ***/
/***  (according to clause 5.5.2)                                                          ***/
/***  Program Version 1.0     Release January 2010                                         ***/
/**********************************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#define MAX_SAMPLES 5000000
#define MAX_LMA 100000

/*** Declare function prototype ***/
int File_info(char *  , double *);
int readcsv(char * ,double * ,double * );
int writecsv(char* ,int ,double * ,double * ,double * );
int Sliding_LMA(double ,int ,double *,double *,int ,double *,double *,double *);
void Noise_STD(int ,double *,double *, double *,double *);


/**********************************************************************************************/
/***    File_info  function                                                                ***/
/***                                                                                       ***/
/***    Description:                                                                       ***/
```

```
/***    This function parse a file in CSV format                                                  ***/
/***    to determine the number of lines and sampling rate                                        ***/
/***                                                                                              ***/
/***    Input: filename                                                                           ***/
/***                                                                                              ***/
/***    Return: Number of samples  (sample Count)                                                 ***/
/***            0  if an error occurred                                                           ***/
/***                                                                                              ***/
/**************************************************************************************************/
int File_info(char * fname , double * pdt)
{
    int i , c;
    double t1,v1,t2,v2;
    FILE     *sample_file;

   /************ Open File **********************/
   if (!strchr(fname, '.'))   strcat(fname, ".csv");

   if ((sample_file = fopen(fname, "r"))== NULL)
   {
      printf("Cannot open input file %s.\n",fname);
      return 0;
    }
    /*** Read two first lines to retrieve sampling rate ***/
    fscanf(sample_file,"%Lf,%Lf\n", &t1, &v1);
    if (feof(sample_file))
    {
        fclose(sample_file);
        return 0;
    }
    fscanf(sample_file,"%Lf,%Lf\n", &t2, &v2);
    if (feof(sample_file))
    {
        fclose(sample_file);
        return 0;
    }
    *pdt=t2-t1;

    i=2;

    while (!feof(sample_file))
    {
        c = fgetc(sample_file);
        if (c == '\n')
        {
            i++;
            if (i>=MAX_SAMPLES)
            {
                printf("Too many samples in input file: only %d samples retained\n",i);
                break;
            }
        }
    }
    fclose(sample_file);
    return i;
}
/**************************************************************************************************/

/**************************************************************************************************/
/***    Read CSV File  Function                                                                   ***/
/***                                                                                              ***/
/***    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, Sampling rate, Max/Min Voltage                                     ***/
/**************************************************************************************************/

int readcsv(char* fname,double vtime[],double vd[])
{
   double max_vd,min_vd;
```



```
   int i;

   FILE     *sample_file;

   /************ Open File **********************/
   if (!strchr(fname, '.'))   strcat(fname, ".csv");

   if ((sample_file = fopen(fname, "r"))== NULL)
      {
             printf("Cannot open input file %s.\n",fname);
             return 0;
      }


/*****************************************************************************************************/
/* Read CSV File                                                                                  ***/
/*****************************************************************************************************/
   max_vd=-1e-9F;
   min_vd=-max_vd;
   i=0;

   while (!feof(sample_file))
      {
        if (i>=MAX_SAMPLES)
        {
        printf("Too many samples in input file: only %d samples read\n",i);
        break;
        }
      fscanf(sample_file,"%lf,%lf\n", &vtime[i], &vd[i]);
      if (vd[i]>max_vd) max_vd=vd[i];
      if (vd[i]<min_vd) min_vd=vd[i];
      i++;
      }
   fclose(sample_file);

   /************ Displays Statistics  ***********************/
   printf("\n**********************************************\n");

   printf("Statistics: \n");
   printf(" Filename    : %s\n",fname);
   printf(" Sample count: %d\n",i);
   printf(" Sampling rate : %1.0f MHz\n",1e-6/(vtime[1]-vtime[0]));
   printf(" Max(vd)     : %4.0f mV\n",max_vd*1000);
   printf(" Min(vd)     : %4.0f mV\n",min_vd*1000);
   return i;
}
/****** End ReadCsv **********************************************************************************/

/*****************************************************************************************************/
/***    Write CSV File  Function                                                                  ***/
/***                                                                                              ***/
/***    Description:                                                                              ***/
/***    This function writes in a CSV format file                                                 ***/
/***    the result of LMA computation:                                                            ***/
/***    Fist column   = time(s)                                                                   ***/
/***    Second column = Upper side band amplitude (V)                                             ***/
/***    Third column  = Lower side band amplitude (V)                                             ***/
/***                                                                                              ***/
/***    Return: Number of written samples                                                         ***/
/***            0  if an error occurred                                                           ***/
/*****************************************************************************************************/

 int writecsv(char* fname,int n_LMA,double LMA_time[],double USB[],double LSB[])
 {
     int i;
     FILE     *out_file;
     if ((out_file = fopen(fname, "w"))== NULL)
      {
             printf("Cannot open output file %s.\n",fname);
             return 0;
      }
     for (i=0;i<n_LMA;i++)
     {
         fprintf(out_file,"%7.4E,%7.4E,%7.4E\n",LMA_time[i],USB[i],LSB[i]);
     }
     fclose(out_file);
     return i;
 }
```

```
/*********************************************************************************************/
/***    Sliding_LMA : Load Modulation Amplitude versus Time                               ***/
/*********************************************************************************************/
/***    Description:                                                                      ***/
/***    This function calculates Upper side band and                                      ***/
/***    Lower side band amplitudes  as a function of time                                 ***/
/***                                                                                      ***/
/***    Arguments:                                                                        ***/
/***    fc = carrier frequency (Hz)                                                       ***/
/***    count = number of input signal samples                                            ***/
/***    vtime[] = input signal time array                                                 ***/
/***    vd[] =   input signal voltage array                                               ***/
/***    lout =   max. size of following arrays                                            ***/
/***    LMA_time[] = Times to which LMAs are computed                                     ***/
/***    USB[] = load modulation amplitude at fc+fs                                        ***/
/***    LSB[] = load modulation amplitude at fc-fs                                        ***/
/***                                                                                      ***/
/***    return value: number of computed LMA                                              ***/
/*********************************************************************************************/
int  Sliding_LMA(double  fc,int  count,double  vtime[],double  vd[],int  lout,double  LMA_time[],double
USB[],double LSB[])
{
    double c1_real,c1_imag;
    double c2_real,c2_imag;
    double w0,wu,wl,dt;
    double Wb;                /* Bartlett window coefficient */
    int i,j,k=0;
    int N_data;              /* Time window size*/
    int N_over;              /* Overlap */
    int N_middle;             /* Half window size */
    double *Yuc,*Yus,*Ylc,*Yls; /* Phase factors */
    double  pi;   /* pi=3.14.... */
    double sum_Wb=0; /* Sum of Bartlett coeff. */
    double cf; /* correction factor of the Bartlett window */

    pi = (double)atan(1.0)*4; /* calculate pi */

    w0=(double)(fc*2.0)*pi; /* carrier 13.56 MHz */
    wu=(double)(1.0+1.0/16.0)*w0; /* upper sideband 14.41 MHz */
    wl=(double)(1.0-1.0/16.0)*w0; /* lower sideband 12.71 MHz */


    /********** Time window ******************************/
    dt=vtime[2]-vtime[1]; /* Note: (vtime[2]-vtime[1]) is the scope sampling rate */
    N_data=(int)(0.5+2*16.0F/dt/fc); /* Number of samples for two subcarrier periods */
    N_middle=(int) (0.5+N_data/2);
    N_over=(int) (0.5+1.0/dt/fc); /* Overlap of 1/fc */

    /******* Allocate memory *****************************/
    Yuc=(double *) malloc(N_data * sizeof(double));
    if (Yuc == NULL)
    {
        printf("Cannot allocate memory");
        return 0;
    }
    Yus=(double *) malloc(N_data * sizeof(double));
    if (Yus == NULL)
    {
        printf("Cannot allocate memory");
        free(Yuc);
        return 0;
    }
    Ylc=(double *) malloc(N_data * sizeof(double));
    if (Ylc == NULL)
    {
        printf("Cannot allocate memory");
        free(Yuc); free(Yus);
        return 0;
    }
    Yls=(double *) malloc(N_data * sizeof(double));
    if (Yls == NULL)
    {
        printf("Cannot allocate memory");
        free(Yuc); free(Yus); free(Ylc);
        return 0;
    }
```



```
/************* Calculate apodization window and phase factors ********/
for( i=0;i<N_data;i++)
    {
        /* Bartlett window */
        if ((N_data & 1) == 0)
        {
            /* N_data is even */
            if (i < (int) (N_data /2))
            {
                Wb=2.0F*i/(double)(N_data - 1);
            }
            else
            {
                Wb=2.0F*(N_data-i-1)/(double)(N_data - 1);
            }
        }
        else
        {
            /*N_data is odd */
            if (i <= (int) (0.001+(N_data-1) /2))
            {
                Wb=2.0F*i/(double)(N_data - 1);
            }
            else
            {
                Wb=2.0F-2.0F*i/(double)(N_data - 1);
            }

        }
        Yuc[i]=(double)cos(wu*i*dt)*Wb;
        Yus[i]=(double)sin(wu*i*dt)*Wb;
        Ylc[i]=(double)cos(wl*i*dt)*Wb;
        Yls[i]=(double)sin(wl*i*dt)*Wb;
        sum_Wb += Wb;
    }
    cf=N_data/sum_Wb;

/****************** DFT ********************************/

for (j=0;j<count-N_data;j=j+N_over)
{
    c1_real=0; /* real part of the up. sideband fourier coefficient */
    c1_imag=0; /* imag part of the up. sideband fourier coefficient */
    c2_real=0; /* real part of the lo. sideband fourier coefficient */
    c2_imag=0; /* imag part of the lo. sideband fourier coefficient */

    for( i=0;i<N_data;i++)
    {
        c1_real=c1_real+vd[i+j]*Yuc[i];
        c1_imag=c1_imag+vd[i+j]*Yus[i];
        c2_real=c2_real+vd[i+j]*Ylc[i];
        c2_imag=c2_imag+vd[i+j]*Yls[i];

    }

    /****************** DFT scale ***************************/

    c1_real=2.0F*cf*c1_real/(double) N_data;
    c1_imag=2.0F*cf*c1_imag/(double) N_data;
    c2_real=2.0F*cf*c2_real/(double) N_data;
    c2_imag=2.0F*cf*c2_imag/(double) N_data;

    /**************** absolute fourier coefficient **********/
    USB[k]=(double)sqrt(c1_real*c1_real + c1_imag*c1_imag);
    LSB[k]=(double)sqrt(c2_real*c2_real+c2_imag*c2_imag);

    /************* Half window time ***********************/
    LMA_time[k]=vtime[j+N_middle];   /* Half window time */
    k++;
    if (k > lout) break; /* stop if array size is reached*/
}

free(Yuc);
free(Yus);
free(Ylc);
free(Yls);
return k;
```

```
}
/*********  End DFT *************************************************************************/

/*******************************************************************************************/
/***    Noise_STD : Noise standard deviation                                             ***/
/*******************************************************************************************/
/***    Description:                                                                     ***/
/***    This function calculates the standard deviations                                 ***/
/***    at fc+fs and fc-fs as required by the noise                                      ***/
/***    precondition test of ISO/IEC 10373-6.                                            ***/
/***    Results are meaningful only when the sense coil's                                ***/
/***    signal is recorded with a reference PICC.                                        ***/
/***                                                                                     ***/
/***    Arguments:                                                                       ***/
/***    n_LMA = number of input values                                                   ***/
/***    USB[] = load modulation amplitude at fc+fs                                       ***/
/***    LSB[] = load modulation amplitude at fc-fs                                       ***/
/***    pSTD_USB= standard deviation at fc+fs                                            ***/
/***    pSTD_LSB= standard deviation at fc-fs                                            ***/
/***                                                                                     ***/
/*******************************************************************************************/
void Noise_STD(int n_LMA,double USB[],double LSB[], double *pSTD_USB,double *pSTD_LSB)
{
    double P_USB=0,P_LSB=0;
    int i;

    /**************** Square summation ********************/
    for( i=0;i<n_LMA;i++)
    {
        P_USB += USB[i]*USB[i];
        P_LSB += LSB[i]*LSB[i];
    }
    *pSTD_USB=sqrt(P_USB/n_LMA);
    *pSTD_LSB=sqrt(P_LSB/n_LMA);

}

/*******************************************************************************************/
/***    MAIN  Program                                                                    ***/
/*******************************************************************************************/
int main(int argc, char *argv[])
{
    char fname[256];
    char fout[256];
    int sample_count;
    int lout;        /*Maximun length of result arrays */
    int n_LMA;          /* Number of computed LMA */
    int status;
    double fc;              /* Carrier frequency */
    double std_USB,std_LSB,dt;
    double *pTime, *pVolts, *pLMA_time, *pUSB, *pLSB;

    fc=13.56e6;

    printf("\n");
    printf("********************************************************************************\n");
    printf("**** ISO/IEC 10373-6 EMD Test-Program   *******************************************\n");
    printf("**** Version: 1.0 January 2010          *******************************************\n");
    printf("****(according to ISO/IEC 10373-6 , clause 5.5.2) *********************************\n");
    printf("********************************************************************************\n");

    if (argc > 1)
    {
        /*** First input parameter is taken as input file name ***/
        strcpy(fname, argv[1]);
    }
    else
    {
        /*** No input parameter ***/
        printf("\nCSV File name :");
        scanf("%s",fname);
    }

    if (!strchr(fname, '.'))   strcat(fname, ".csv");
    if (!(sample_count=File_info(fname , &dt))) return 0;
    lout= (int) (sample_count/(int)(0.5+1.0/dt/fc));
    if (lout > MAX_LMA) lout = MAX_LMA;
```



```
    /***           Start of memory allocation                ***/
    pTime= (double *) malloc(sample_count * sizeof(double));
    if (pTime == NULL)
    {
        printf("Cannot allocate memory");
        return 0;
    }
    pVolts= (double *) malloc(sample_count * sizeof(double));
    if (pVolts == NULL)
    {
        printf("Cannot allocate memory");
        free(pTime);
        return 0;
    }
    pUSB= (double *) malloc(lout * sizeof(double));
    if (pUSB == NULL)
    {
        printf("Cannot allocate memory");
        free(pTime); free(pVolts);
        return 0;
    }
    pLSB= (double *) malloc(lout * sizeof(double));
    if (pLSB == NULL)
    {
        printf("Cannot allocate memory");
        free(pTime); free(pVolts); free(pUSB);
        return 0;
    }
    pLMA_time= (double *) malloc(lout * sizeof(double));
    if (pLMA_time == NULL)
    {
        printf("Cannot allocate memory");
        free(pTime); free(pVolts); free(pUSB); free(pLSB);
        return 0;
    }
    /***         End of memory allocation        ***/

    if (!(sample_count=readcsv(fname,pTime,pVolts))) return 0; /* reading data */
    if (!(n_LMA=Sliding_LMA(fc,sample_count,pTime,pVolts,lout,pLMA_time,pUSB,pLSB))) return 0;

       /***** processing data ***********************************/
    strcpy(fout,"LMA_");
    strcat(fout,fname);
    status=writecsv(fout,n_LMA,pLMA_time,pUSB,pLSB); /* writing results in a file */

    Noise_STD(n_LMA,pUSB,pLSB,&std_USB,&std_LSB);  /* evaluating noise floor */

    /************** Result Display *************************/
    printf("\n");
    printf("**********************************************\n");
    printf(" Noise floor : \n");
    printf(" standard deviation at fc+fs: %7.3f mV\n",std_USB*1000);
    printf(" standard deviation at fc-fs: %7.3f mV\n",std_LSB*1000);
    printf(" Note: Displayed results are meaningful only when\n");
    printf("      the sense coil's signal is recorded with a\n");
    printf("      reference PICC.");
    printf("\n**********************************************\n");

    free(pTime);
    free(pVolts);
    free(pLMA_time);
    free(pUSB);
    free(pLSB);

    return 1;
}
/*********** End Main  ***************************************************************************/
```