THE SOUND CARD PROGRAMMING?...BUT IT IS EASY!

F6CTE the 21/12/2005 (with Bill KA0VXK for English corrections)

In this paper, I present a DLL (a program able to be loaded and used by other programs) allowing to consider the sound card as a "black box". This could interest the ones for which the Windows API programming is a problem and who would wish to use the sound card for experimentation, as, for example, decoding AF receivers.
The only need is that the DLL and the program be in the same directory.

F6CTE_DLL_RX_SOUND_CARD_V1 DLL

SUMMARY

1) ANNOUNCEMENT OF PRESENTATION OF THE DLL F6CTE_DLL_RX_SOUND_CARD_V1 AND TYPE OF USE

Goal of this DLL

I have made available to all, a DLL and test programs (downloadable from http://f6cte.free.fr) allowing one to easily access radio programs based on the sound card.
This DLL is a freeware, for non-commercial use only. This program has a "bugs" risk.

The "painful" part of the sound card processing is handled by the DLL so it is transparent to the user.

Now the programming: it should be considered that SDR (Software Defined Radio) looks like electronic tubes or the electronic field effect (better) for which the input impedances are infinite. On the other hand, the output impedances are equal to 0 ohm.So, there is no matching to do.

The amplifications are ideal and perfectly linear:
B (level after amplification) = K(amplification factor) x A (level before amplification) strictly.

However, one must take into account the sampling precision of 8 bits: a digital signal can take only one level among 256, when an analog signal has an infinite amount of levels.

If all of this is taken account, all the AF schematics are able to be reduced to software. For example, I use Costas loops, mixers.... The computerized receivers (the ones of Multipsk for example) are comparable to electronic receivers with a big diversity of possibilities as in analog electronic. For example, there are many ways to decode PSK31:

* according to the way to position a bandpass filter ahead or a "complex" lowpass filter on the quadrature components of the baseband signal,
* according to the way to process phase errors for the AFC (Automatic Frequency Control)...

To be exact however, there are algorithms which take advantage of the PC computing power to accelerate the calculations or the convergences, or to do too much complex processings for analog electronic (as FFT) but, at first glance, it must be possible to use all classical electronic formulas:

* if the worked frequency is much lower than the sampling frequency,
* as long as they are simple operations (+, -, /, x).

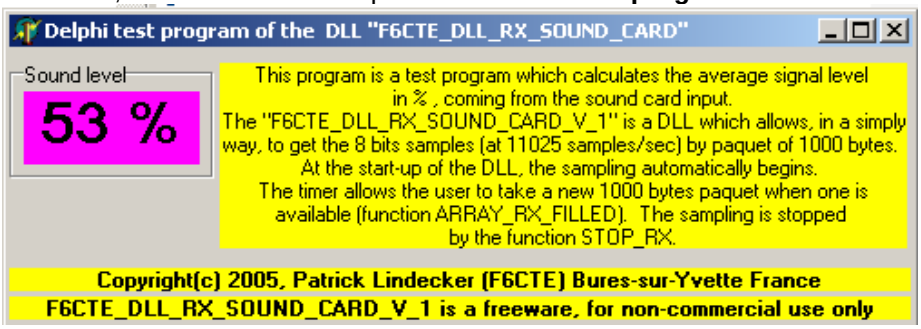
Now about the filtering, where it is a convolution operation (between an input e(t) and an impulsional filter response h(t))), things are a bit more complex (see the example of the lowpass filter at the chapter 3).

There are several peculiarities due to the fact that the sampling frequency is not infinite (but equal to 11025 samples/sec for this DLL), which complicates things a bit. There are, however, many books about this subject (signal processing and transmissions), unfortunately a bit complicated (whereas the practice is not so complicated).

With the present sound cards, one cannot process frequencies higher than 20000 Hz (the maximum sampling frequency being of 44100 samples/s) but the principles being the same, nothing, will prevent the processing of signals at a higher frequency, in the same way, if, some day, HF cards were proposed, as standard, on the PC.

2) DESCRIPTION OF A TEST PROGRAM

Hereafter, it will be found a snapshot of the **first test program.**



For information, this is the **main part of the first test program (in Pascal/Delphi 6)** allowing the average level of the sound card input AF signal. Comments are between {}.

```
procedure TForm1.Timer1Timer(Sender: TObject);
VAR COUNTER:LONGINT;
VAR LEVEL:BYTE;{between 0 to 255}
VAR DATA:LONGINT;{between -255 to 255}
VAR SUM:LONGINT;
VAR AVERAGE:LONGINT;{between 0 to 100}
VAR STRING_AVERAGE:STRING;
begin
 ARRAY_FILLED:=ARRAY_RX_FILLED__F6CTE_DLL_RX_SOUND_CARD_DELPHI_V_1(ARRAY_RX);
 {a data string is ready to be managed}
 IF ARRAY_FILLED=TRUE THEN
 BEGIN
  {the average level is computed and then displayed}
  SUM:=0;{initialization}
  {for the oldest (COUNTER:=0) to the newest (COUNTER:=255)}
  FOR COUNTER:=0 TO 999 DO
  BEGIN
   LEVEL:=ARRAY_RX[COUNTER];
   DATA:=2*LEVEL-255;
   INC(SUM,ABS(DATA));
  END;

  {calculation of the average}
  AVERAGE:=SUM DIV 1000;
  {normalization of the average to 100 %}
  AVERAGE:=(AVERAGE * 100) DIV 255;

  {display of the average}
  STR(AVERAGE,STRING_AVERAGE);
  The_Level.Caption:=STRING_AVERAGE+' %';
 END;
end;
```

Some explanations about this program

Timer1Timer is a procedure which is started up periodically (every 10 ms here) by the Windows clock.

It is tested if a complete data array (a set of data under the form of bytes) has been taken by the DLL. If so, it is returned the flag ARRAY_FILLED:=TRUE, conversely ARRAY_FILLED:=FALSE.

If the array is ready (ARRAY_FILLED=TRUE), the array is processed with the "FOR COUNTER:=0 TO 999 DO" loop.

Each of these data array samples is taken and normalized to -255 à 255:
LEVEL:=ARRAY_RX[COUNTER];
DATA:=2*LEVEL-255;

It is, afterwards, done the sum of the absolute values of the sample levels:
INC(SUM,ABS(DATA));

It is computed the average and it is normalized to 0 - 100 %:
AVERAGE:=SUM DIV 1000;
AVERAGE:=(AVERAGE * 100) DIV 255;

The digital average is transformed in string of characters:
STR(AVERAGE,STRING_AVERAGE);

This average is displayed:
The_Level.Caption:=STRING_AVERAGE+' %';

Some answers to possible questions
The array can be fixed or not. The user does what he wants, but a problem of too intense a load can occur on a slow PC, if the array is reduced.

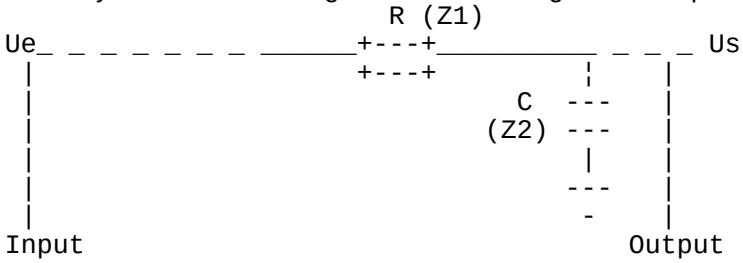
The samples are only done in 8 bits and the sampling frequency is fixed at 11025 samples/second.

The DLL code is not diffused (it is simply a "black box" for the users). The only code available is formed by the test programs. The necessary explanations are given by the comments on the test programs and by this document.

3) CALCULATION OF A DIGITAL LOWPASS FILTER, RECURSIVE TYPE, USED AS TEST number 2 IN THE DLL F6CTE_DLL_RX_SOUND_CARD_V1

A simple example shows how to digitally simulate a lowpass filter with a recursive type algorithm.

Let's say that we want to digitalize the following first order passive lowpass filter:



(signal ~ of pulsation w=2.Pi.f)

The transfer function in alternative is:
H(w)=Z2/(Z1+Z2)=(1/jCw)/(R+(1/jCw))=(1/jCw)/((RjCw+1)/jCw))=1/(1+jRCw)
Note: the square of the module of this function is: H.H*=1/(1+RCw*RCw)

Let's say: p=jw (Laplace) and T("Tau") =RC, it will be obtained H(p)=1/(1+Tp)

Note: the impulse response h(t) is obtained by reverse Laplace transform. It will be found: h(t)=(1/RC) x exp(-t/RC)

It is more practical to work on analog signal sampling to pass in discrete signal by the Z transform (Z=exp(p.Te)).
Te = 1/Fe with Fe the sampling frequency

Assimilating p to (1-Z')/Te (Euler approach) with the sampling period =1/Fe and Z' meaning Z to the power -1, it will be obtained:
H(Z):=1/(1+T(1-Z')/Te)=Te/(Te+T-TZ')
S(Z) = H(Z).E(Z), with S(Z) the Z output and E(Z) the Z input.
So S(Z).(Te+T-TZ')=E(Z).Te
S(Z)(Te+T)-S(Z).T.Z'=E(Z).Te

By Z reverse transform, it will be found:
s(n).(Te+T)-s(n-1).T=e(n).Te
So s(n) = s(n-1).T/(Te+T) + e(n).Te/(Te+T)

Clearly, the output at the moment t depends on the output at the moment t-Te (previous output) and on the input sample at the moment t.

Application: suppose that R=1000 ohm, C=0,159 µF, which theoritically gives a cut-off frequency fc at à - 3 dB of 1000 Hz (fc= 1/(2.Pi.R.C)), for a phase shift of - 45 °.
With these hypothesis, it will be found T=RC=0.159 ms
If the sampling frequency is Fe=11025 samples/sec (Fe of the DLL), Te=1/Fe is worth 0.0907 ms and it is obtained the formula:
s(n) = s(n-1).0.637 + e(n).0.363

After test, it will be found output mudules S close to the following theoritical levels:

```
* F =  100 Hz  S=99 %
* F =  300 Hz  S=96 %
* F =  500 Hz  S=89 %
* F =  800 Hz  S=78 %
* F = 1000 Hz  S=71 %
* F = 1200 Hz  S=64 %
* F = 1500 Hz  S=55 %
* F = 2000 Hz  S=45 %
```

Note: theoretically, there is nothing to prevent the direct application of the convolution operation (e(t) * h(t)) in the temporal domain (without using the previously described method), if this operation is done on a sufficient number of samples ("taps"). It is reminded that the convolution operation is decomposable in elementary operations (+ and X).

4) LIST OF THE TEST PROGRAMS

- There will be found only one test program in C++ (TEST_F6CTE_DLL_RX_SOUND_CARD_C_V_1.exe), the one relative to the calculation of the average level of the input signal.

- In Pascal (Delphi 6), it will be found the same program (TEST_F6CTE_DLL_RX_SOUND_CARD_D_V_1.exe) plus the lowpass filter (TEST2_F6CTE_DLL_RX_SOUND_CARD_D_V_1.exe) which has been presented in chapter 2. By pushing on the "Filter" button, the filter is inserted, and conversely. The output is an average level.