Using Matlab/Simulink as an implementation tool for Multi-Channel Surround Sound

P. Schillebeeckx, I. Paterson-Stephens, B. Wiggins
Signal Processing Applications Research Group, University of Derby
Derby, DE22 1GB, United Kingdom

This paper discusses a highly flexible and powerful implementation toolkit that allows new ideas in the field of multi-channel surround sound to be implemented with great ease and minimal development time. The key feature of the toolkit is that it enables complex surround sound encoding and decoding algorithms to be implemented in real-time with relative ease.

Introduction

The authors experience in designing and implementing encoding and decoding systems for multi-channel surround sound showed that although the design of such systems was not in itself excessively complex, the time consuming nature of designing circuit boards and software routines for each new system, often proved to be prohibitive. This was especially true if a quick test of a new idea or innovation was required.

The availability of a rapid development and prototyping toolkit for the design and implementation of multi-channel surround sound processors soon became a clear requirement. This paper describes a highly flexible software based toolkit, designed to operate on a suitably equipped Pentium PC. The software toolkit enables up to 16-channels of 16 bit 48k audio to be processed in real-time. The toolkit also provides a collection of processing blocks specific to multi-channel sound processing.

The Aim

The key aim of the project was to produce a highly flexible software toolkit, which can be re-configured quickly an easily with minimal effort and cost.

High flexibility should mean that the toolkit can be used to implement a wide variety of surround sound systems, ranging from two channel (such as dipole and binaural) through to the 3-2 consumer formats (more commonly known as 5.1) and higher-order ambisonics systems. With this level of flexibility in mind, the software toolkit incorporates a series of software routines 'building blocks' to perform operations such as: Signal Matrixing, Complex filtering (for HRTF, Cross-talk cancellation and room response characterisation), Fast convolution, Fourier and Hilbert transformations, Variable delays, Pan-rotate-tilt and Multi-channel I/O.

The system had to be low cost (doesn't it always) which in the authors case meant that wherever possible the system should make use of existing resources. In practical terms this meant that the use of a standard Pentium PC equipped with a multi-channel sound card was the obvious choice.

System Overview

As already mentioned, a standard PC forms the basic hardware platform for the system. The PC needs to be equipped with a multi-channel sound card (Soundscape Mixtreme in the authors case), see figure 1. The software toolkit was designed to operate within Simulink, which, in its basic form, is a block-diagram based approach to the Matlab programming language.

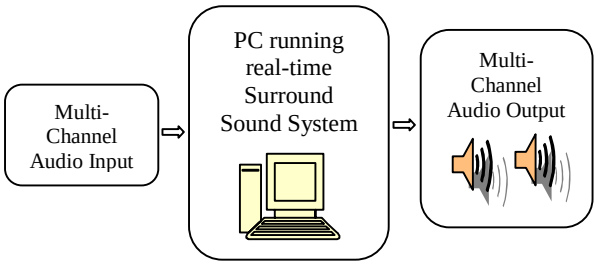

Fig. 1. Overview of the system.

Introduction to Simulink:

Simulink incorporates a wide variety of building blocks, which can be used to develop complex simulations of a wide range of systems, such as mechanical vibration models and complex electronic control systems.

Using Simulink, it is possible to simulate continuous, discrete and hybrid systems using varied solver methods as required to meet the needs of a specific application [1, Simulink].

For the purpose of the system described in this paper the following additional Simulink toolboxes were required: DSP Blockset, Fixed-Point Blockset and Real-Time Workshop. These standard blocksets add various discrete time operations to the basic Simulink application as well as the ability to translate software designs to work on other platforms [2, RTW].

Input and Output Interface:

Although Simulink incorporates I/O blocks which can be used to interface a software model to a standard PC soundcard, none are suitable to deal with multi-channel surround sound in their existing form. Multiple instances of the two-channel audio I/O block, which is provided as standard within Simulink, could in principle be used so that, for example, 16 channels of audio could be processed simultaneously. The problem with this is that variable delays exist between blocks, as the Windows operating system is not capable of opening more than one audio device instantaneously, see figure 2.

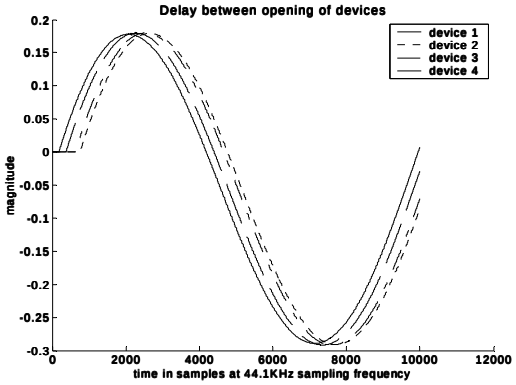

Fig. 2. Delay between audio devices

The first challenge in using Simulink for multi-channel audio processing is therefore to produce a block capable of opening multiple audio channels simultaneously. By re-coding parts of Simulink, it was possible to produce a set of audio I/O blocks capable of handling eight and sixteen channels. These interface directly with the soundcard and do not suffer any of the latency and synchronisation problems previously experienced. In order to overcome the problem of opening multiple devices instantaneously, the audio device has been set to operate as a single sixteen-channel device. In this case whenever the audio device is opened, all sixteen channels will be perfectly synchronised.

Case Study: An 8-Channel Ambisonics Decoder

To demonstrate the application of the Multi-channel sound Simulink toolbox, a case study involving the design and real-time implementation of an 8-channel ambisonics decoder will be considered. The implementation of an 8-Channel Ambisonics Decoder relies on basic sine and cosine laws and a set of signal matrixing operations.

Ambisonics – *an overview*:

Ambisonics [3, Gerzon; 4, Gerzon] is a surround system based on the decomposition of a soundfield using spherical harmonics, see figure 3. The spherical harmonics contain a zero'th order omni-directional pressure signal (W) and three directional particle velocity signals (X,Y,Z), front-back, left-right and up-down.

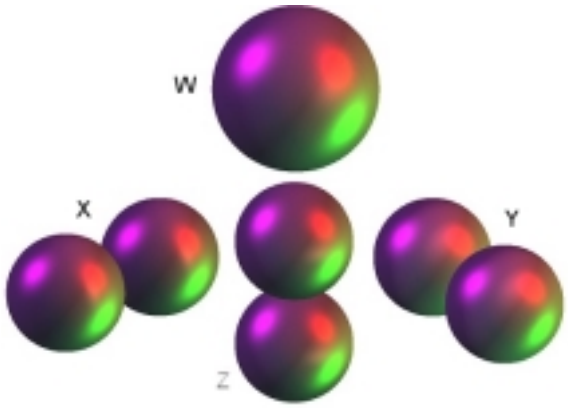

Fig. 3. Spherical harmonics forming B-format

The W, X, Y and Z signals are jointly referred to as the B-format signal and together these are used to describe a full-periphonic soundfield. From these signals it is possible to derive a loudspeaker feed for any position on the unit sphere. Such a loudspeaker feed is dictated by the following relationship:

$$\begin{aligned} LSP\_feed(azimuth, elevation) &= \sqrt{(2)} \times W \\ &+ \cos(azimuth) \times \cos(elevation) \times X \\ &+ \sin(azimuth) \times \cos(elevation) \times Y \\ &+ \sin(elevation) \times Z \end{aligned}$$

For this particular case study, the B-Format signal will be decoded to 8 speaker feeds, each equally spaced apart (45 degrees azimuth) at 0 degrees elevation, see figure 4.

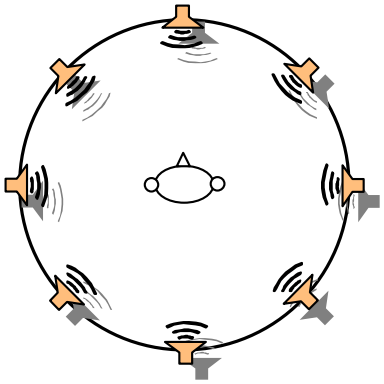

Fig. 4.Loudspeaker layout

Using a package such as MATLAB, it is possible to create a mathematical model or even process B-format audio files off-line [5, MATLAB], resulting in the required speaker feeds. This will allow for a mathematical verification of the theory, but does not allow for any real-time adjustments. These real-time adjustments are particularly useful in surround sound systems, as the final tweaks are often down to subjective fine-tuning.

Non Real-Time model using MATLAB:

The program below shows the mathematical principles behind the decoder, any B-format signal can be used, both recorded or simulated:

```
%  An off-line ambisonics decoder for an 8-
%  channel loudspeaker set-up. The height
%  factor is ignored as all speakers have an
%  elevation of 0.

%  B-format input (simulated or recorded B-
%  format)
W= a signal;
X= a signal;
Y= a signal;

%  Declarations of the loudspeaker positions
azim_0   = 0;
azim_45  = pi/4;
azim_90  = pi/2;
azim_135 = 3*pi/4;
azim_180 = pi;
azim_225 = 5*pi/4;
azim_270 = 3*pi/2;
azim_315 = 7*pi/4;

%  Calculation of the loudspeaker feeds
LSP_0   = sqrt(2)*W + cos(azim_0)*X
                          + sin(azim_0)*Y;
LSP_45  = sqrt(2)*W + cos(azim_45)*X
                          + sin(azim_45)*Y;
LSP_90  = sqrt(2)*W + cos(azim_90)*X
                          + sin(azim_90)*Y;
LSP_135 = sqrt(2)*W + cos(azim_135)*X
                          + sin(azim_135)*Y;
LSP_180 = sqrt(2)*W + cos(azim_180)*X
                          + sin(azim_180)*Y;
LSP_225 = sqrt(2)*W + cos(azim_225)*X
                          + sin(azim_225)*Y;
LSP_270 = sqrt(2)*W + cos(azim_270)*X
                          + sin(azim_270)*Y;
LSP_315 = sqrt(2)*W + cos(azim_315)*X
                          + sin(azim_315)*Y;
```

Note that the elevation factor and it's relevant spherical harmonic Z have been discarded from the above program, as all speakers are at 0 degrees elevation.

A more useful model would be one capable of simulating the system in real-time allowing for real-time parametric adjustment, giving a far more flexible method of adjusting and analysing system performance.

Real-Time implementation using Simulink:

For real-time operation, Simulink and an associated multi-channel sound toolkit has been developed. As mentioned earlier, Simulink allows a block diagram approach to programming, as will be demonstrated.

The 8-channel ambisonics decoder, this time using basic Simulink Building blocks, is shown in Fig. 5 at the end of this paper.

Creation of dedicated processing blocks:

Once a system or part of a system has been designed successfully, it can be converted into a building block and added to the library of parts for future use. The simple 8-channel ambisonics decoder shown in figure 5, although not highly complex is already somewhat of a maze, and adding more functionality to the system will clutter the view further. The ability to *contain* sub-systems within individual building blocks allows for logical and transparent designs to be built up.

In the software toolkit, the 8-channel ambisonics decoder appears as a single building block which can be added to any system. The decoder can then be used as the output stage of more complex surround sound system designs.

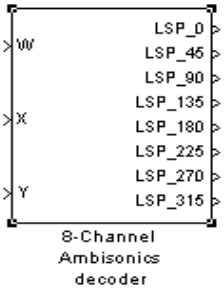

Fig. 6. Eight channel ambisonics decoder block

By specifying the correct I/O interface within the software, the 8-channel ambisonics decoder can be run in real-time.

A fully working 8-channel ambisonics decoder with an audio I/O interface is shown in figure 7. The B-format input signal is fed to the computer, via the soundcard, and processed in real-time. The processed outputs are then fed, again via the soundcard, to eight amplifiers, which in turn feed the loudspeaker array.

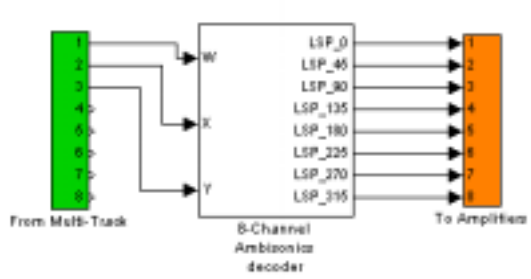

Fig. 7. Decoder with I/O interface

Although the great advantages of real-time modeling will be apparent from the above model, its use is most advantageous in less static models where the real-time control of parameters is of great use.

Figure 8 shows another example of a more dynamic model, which allows for rotation of the soundfield about the Z-axis i.e. a Z-pan control. As before, the input to the system is the standard B-format signal, but rather than just decoding the signal via the eight-channel ambisonics decoder, it is possible to rotate the signal by any angle about the Z-axis.

As W is an omni-directional signal it will be unaltered, X and Y will be changed according to the following rules [6, Malham]:

$$X' = \cos(\alpha) \times X - \sin(\alpha) \times Y$$
$$Y' = \sin(\alpha) \times X + \cos(\alpha) \times Y$$

Where α is the angle of rotation.

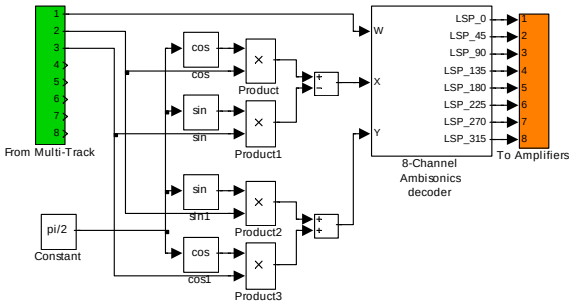

Fig. 8. Soundfield auto rotation

When running the above model it now becomes possible to update the angle of rotation whilst the model is running. Alternatively, the model could be updated automatically, by replacing the constant by a sine wave generator, resulting in a spinning soundfield – autopan effect. Also, the pan/auto-pan control can be placed into a building block and added to the library of parts for future use.

The eight-channel ambisonics decoder is only one of a wide variety of dedicated processing blocks that have been designed. An entire surround sound toolkit has been created containing various encoders, decoders and converters. These include dipole, binaural, ambisonics (first and second order) and 3-2 consumer format encoders and decoders; B-format to 3-2 consumer format, B-format to binaural, B-format to dipole and A-format to B-Format converters etc.

Toolkit Performance

The overall performance of a PC-based real-time toolkit is dependent on several factors, these factors can be categorized into Hardware and Software related issues.

Hardware issues:

The performance of the PC is an obvious factor and is dependent on several issues, although these will not be discussed in this paper. However, it is clear that CPU speed ultimately determines the maximum complexity of model that can be successfully simulated in real-time. The authors experience with this toolkit has been gained using a Pentium III – 600 MHz processor and this has proven to be quite acceptable for most applications.

Being an audio implementation tool, the performance of the multi-channel soundcard is of great importance. Although standard desktop personal computers are not the most ideal places to host audio equipment, many of today's higher-end soundcards perform very well, especially those with external A/D and D/A converters, such as the 16-channel soundscape mixtreme card used in this project.

The soundcard used in this project was capable of sampling frequencies up to 48KHz at 16-bit resolution, for all sixteen input and output channels. Some test results for a basic thru application were carried out (soundcard in -> software mixer -> Simulink model -> software mixer -> soundcard out). The tests included: frequency response, quasi peak rectified noise measurement and total harmonic distortion + noise, these results are shown in figures 9, 10 and 11 respectively. For the purpose of this project the results obtained were deemed to be acceptable.

Software issues:

The accuracy of calculations, and particularly the handling of intermediate results, is an important performance issue when considering the overall performance of the real-time software. The great advantage of using a package such as Simulink is that all calculations are carried out using floating-point

arithmetic. However, if the final target system is to be based on a fixed-point DSP processor for example, then Simulink can configured to perform all calculations using fixed-point arithmetic. This way, the final system performance can be evaluated.

In the case of audio manipulation a fixed-step solver at a set sampling frequency is used. This approach to processing data is analogous to the approach used in DSP algorithms [3, Paterson-Stephens]. All data to be manipulated is arranged in buffers, allowing for efficient data manipulation, the size of these buffers are user defined and can be increased for demanding implementations. An increase in buffer size does however increase the latency of the system.

Latency of the system:

The overall latency of the system is dependant on several factors; some of these factors are determined by the I/O configuration, others by the processes employed.

When configuring the I/O of the system three main parameters are set, the buffer size, the queue duration and the initial output delay [7, DSP Blockset].

The buffer size determines the size of the data buffer passed to the Simulink model and will be used throughout the model.

The queue duration determines the maximum length of time that data acquisition within Simulink can lag data acquisition by the hardware. Also there is a similar parameter to determine the maximum amount of time that the data output from Simulink can lag the data output of the hardware. The maximum queue duration is specified by 1024-buffered audio frames or the size of the hardware buffer on the soundcard whichever is the smallest.

The initial output delay determines the amount of time by which the initial output to the audio device has been delayed.

These I/O parameters should be configured taking into account the likely demands put on the Simulink model, for simple models it will be possible to keep the latency inherent to the I/O interface to a minimum, more complex models may require the latency time to be increased to guarantee stable operation of the model.

The overall latency of the system will consist of the latency inherent to the I/O interface plus the latency due to the processes taking place within the Simulink model.

Conclusions

A flexible real-time simulation and implementation tool, demanding a minimal amount of implementation time has been outlined, with the focus on surround sound systems.

It should be stressed that this is not a tool meant to create finalised products, but a platform upon which to build and fine tune ideas and concepts in the field of multi-channel surround sound. Further, a great advantage of the way the code is structured in Simulink is that it is analogous to the code structure that would be required for DSP routines, allowing for easy porting of code to dedicated DSP processors. Recently toolboxes have been added that will automatically port Simulink simulated systems to specific DSP platforms.

This portability makes the system described a very powerful development tool and a crucial part of a three stage development cycle.

Once the theory for a system has been established, a real-time model can be built up to verify and fine tune the system followed by the final stage, porting the system established onto the DSP platform of choice.

Not mentioned in this paper is the capability of Simulink in combination with the Real Time Workshop blockset to compile Simulink models as stand-alone PC based applications. This ensures the portability of simulations to PC's without any specific software requirements.

The eight-channel ambisonics decoder discussed in this paper, can be downloaded as a stand-alone application from http://sparg.derby.ac.uk.

References

[1] The Math Works Inc, Simulink, Dynamic System Simulation for MATLAB, version 4, 2000.
[2] The Math Works Inc, Real Time Workshop Blockset for Simulink, version 4, 2000.
[3] M.A. Gerzon, "Psychoacoustic Decoders for Multispeaker Stereo and Surround Sound", Proceedings of the 93rd AES Convention, San Francisco 1992.
[4] M.A. Gerzon, General Metatheory of Auditory Localisation, Proceedings of the 92nd AES Convention, Vienna 1992.
[5] The Math Works Inc, MATLAB, version 6, 2000.
[6] D.G. Malham, Spatial hearing mechanisms and sound reproduction, University of York, 1998. http://www.york.ac.uk/inst/mustech/3d_audio/ambis2.htm.

[7] I. Paterson-Stephens, A. Bateman, The DSP Handbook, *Algorithms, Applications and Design Techniques*, Prentice Hall, 2001.
[8] DSP Blockset for Simulink, version 4, 2000, The Math Works Inc.

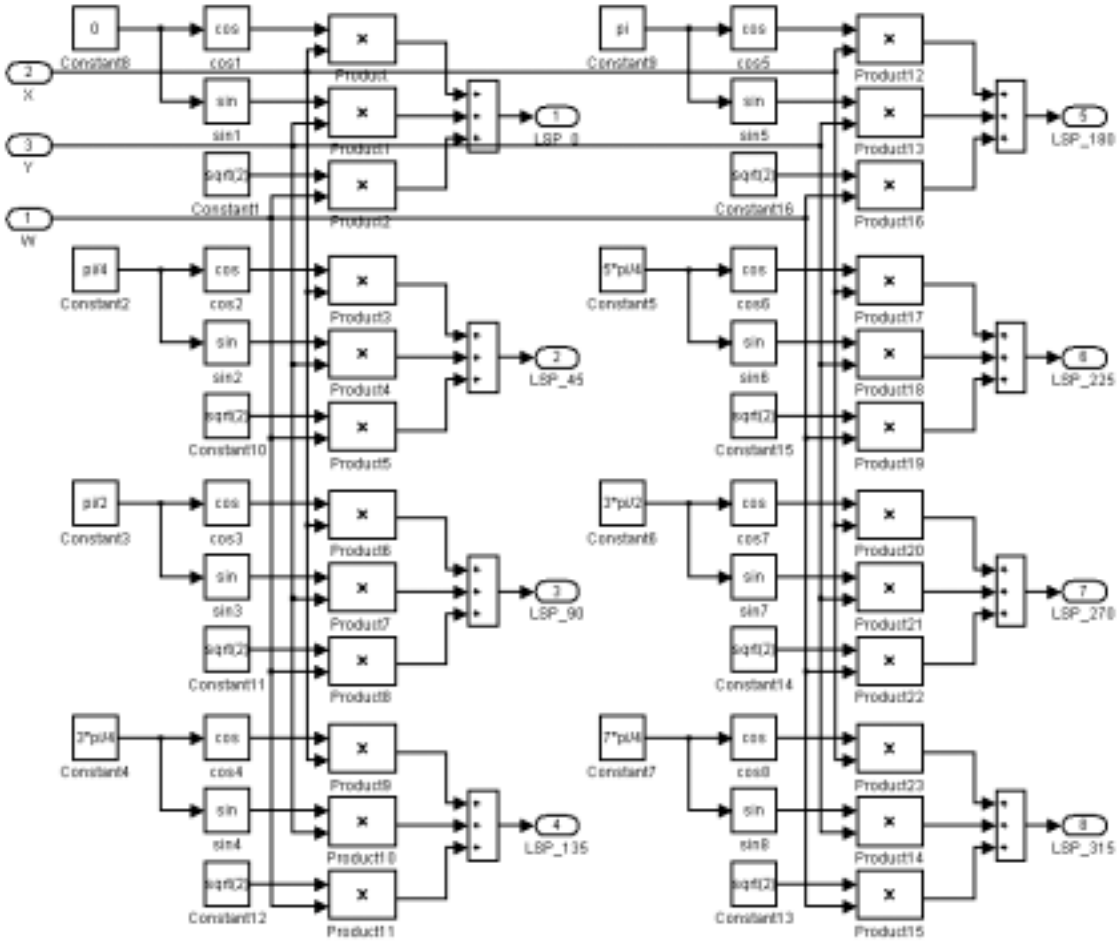

Fig. 5. Eight channel ambisonics decoder using Simulink building blocks.

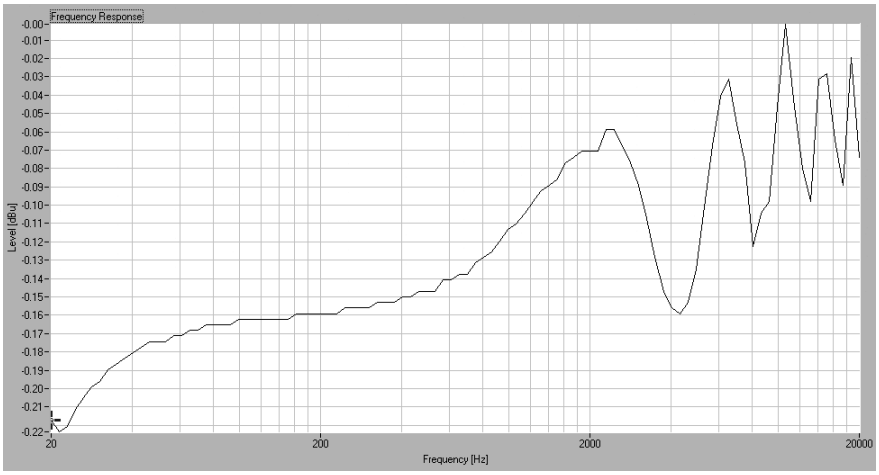

Fig. 9.Frequency response of the system

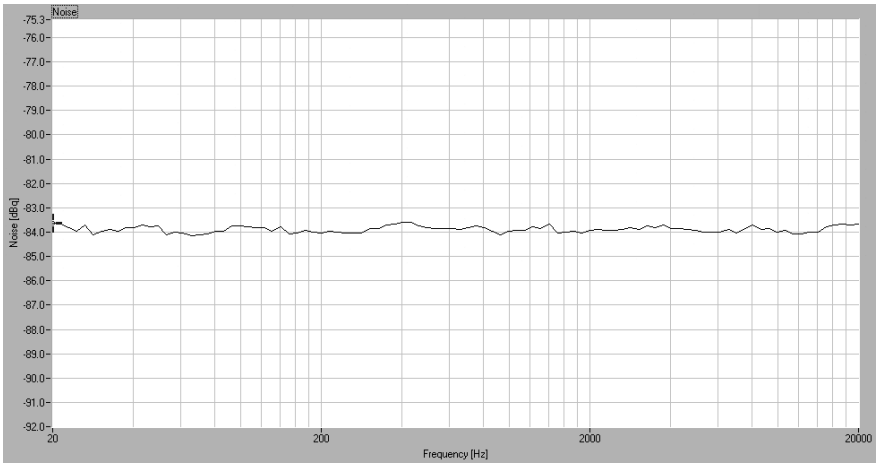

Fig. 10. Quasi peak rectified noise measurement of the system

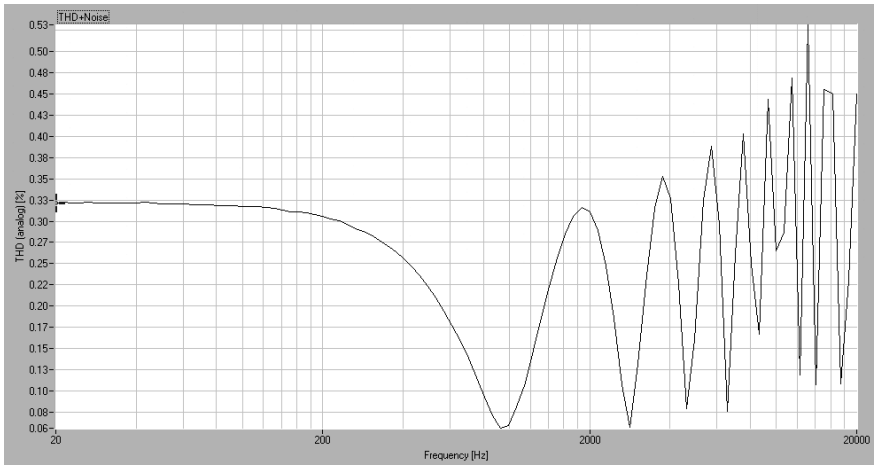

Fig. 11.THD + noise measurement of the system