MODELLING DIGITAL MUSICAL EFFECTS FOR SIGNAL PROCESSORS, BASED ON REAL EFFECT MANIFESTATION ANALYSIS

David Oboril, Miroslav Balik, Jiri Schimmel, Zdenek Smekal, Petr Krkavec

Department of Telecommunications
FEECS, Brno University of Technology
`utko@fee.vutbr.cz`

ABSTRACT

For quite some time in the area of commercial utilization of digital audio effects, efforts have emerged to create simulate by software analog effects and effect processors This paper deals with the analysis of musical effects, the design of algorithms for simulating these effects, and their realization on both digital signal processors and the PC platform in the form of plug-in modules for the DirectX environment. It also deals with the problem of controlling the effect parameters and with subjective testing of algorithms, and it examines the fidelity of simulated effects as compared with the original.

1. INTRODUCTION

In our work we have concentrated on the analysis and simulation of the most common types of analog effect such as parametric equalizer, modulation effects, delay effects, dynamic effects, and effects based on spectral properties of a system with nonlinear transfer characteristic (distortion effects, limiters).

The algorithms of these musical effects are mostly known at a certain accuracy level of models; our main effort was to optimize these algorithms from the viewpoint of implementation on digital signal processors and on the PC platform by the method of plug-in modules or also to find new algorithms by way of analyzing actual analog effects that would better suit the rapid digital processing of signals.

An analysis of analog musical effects depends on the required degree of fidelity of imitation of the musical effect under analysis. The analysis can be performed by monitoring the analyzed musical effect behavior in the time and the frequency domain, or by direct electrical circuit simulation using some simulation software, e.g. PSpice or Matlab. In many cases, following the static features of the effect is enough, however, in some cases, e.g. when we simulate tube amplifiers, we have to start from the system's circuit equations.

When we implement algorithms, we have to solve many problems. Often we have to choose between a precise but time-consuming algorithm or a simpler and fast but less precise algorithm. However, the subjective perception tests, which are the most important aspect in quality evaluation of the simulation, are decisive.

2. DESIGN OF DIGITAL EFFECT ALGORITHMS

Many publications deal with the design of digital musical effect algorithms and forms of their realization. So we will not deal with them. In first part of this paper we will focus on algorithm simulation and adaptation that are necessary for the implementation of effect on DSP or by the Plug-In method. In the second part we will focus on the implementation of chosen digital musical effects.

2.1. Parametric equaliser

Parametric filter structures allow direct access to the parameters of the transfer function, such as gain, bandwidth and centre or cut-off frequency, via the control of associated coefficients. To modify one of these parameters, it is therefore no longer necessary to compute a complete set of coefficients for second order transfer function, bud instead only one coefficient in the filter structure is calculated. An independent control of gain, cot-off/centre frequency and bandwidth is achieved by a feed forward structure for boost and a feed backward structure for cut [1].

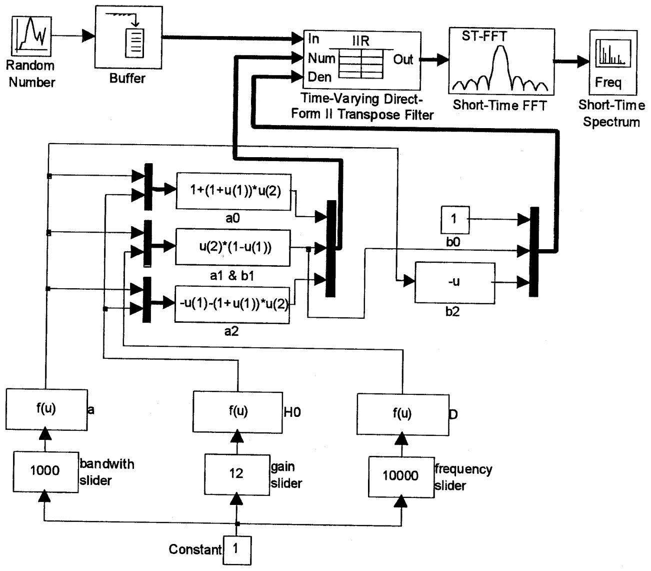

Figure 1.: *Simulink model of Parametric equaliser*

Fig. 1 shows the model prepared for the Simulink program, which was used to simulate the changes in peak filter parameters in real time. White noise is admitted into the time-varying IIR filter. The coefficients of this filter are computed on the basis of slider settings that represent individual filter parameters. Filter behaviour was checked by means of short-term spectral analysis of input signal. Some details can be find in [3].

2.2. Musical Effects with Delay Line

Simple delay effects, such as delay, echo, and multitap echo uses the delay line with constant delay time, while modulation effects, such as phase-shifter, flanger and chorus, use the delay line with variable delay time controlled by low-frequency oscillator (LFO). The block diagram of a general modulation effect for the Simulink program is in Fig. 2.

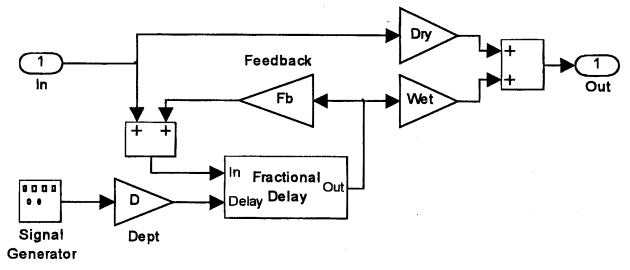

Figure 2: *Simulink model of modulation effects*

The sum of the original (Dry) and the effect (Wet) signal gives the output signal. The delay time of the delay line is changed by the LFO. The block diagram in Fig. 2 corresponds to the flanger effect. When we disconnect the feedback (Feedback=0), the chorus or the phase-shifter effect is produced. The difference between these two effects is only in the delay time of the delay line – the phase-shifter has a much shorter delay time than the chorus. Some details about these effects can be found in [4].

2.3. Effects Based on Transfer Characteristic Non-linearity

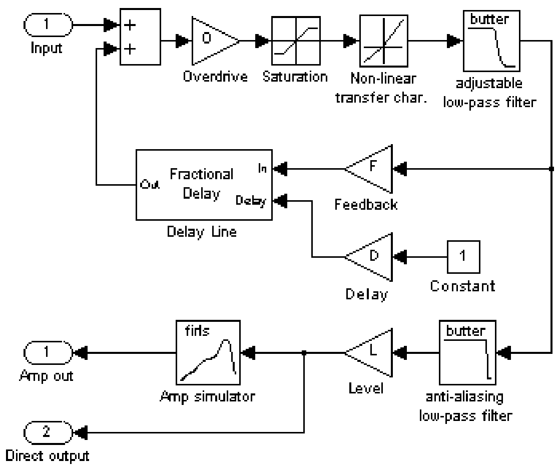

Figure 3: *Simulink model of effects based on transfer characteristic non-linearity*

When we simulated effects based on spectrum modification caused by signal passage through a system with non-linear transfer characteristic, we do not result from system circuit equations but from simulation of individual block of the system – linear amplifier, non-linear component with transfer characteristic that can be approximated using power polynomial and pointed line, filter that simulates linear distortion of system, positive feedback with specific delay, and block that simulates frequency characteristic of guitar combo. General block diagram of the effect based on transfer characteristic non-linearity is in Fig. 3. The effect can simulate tube amplifiers, distortions, etc.

3. REDUCTION AND LINKING OF ALGORITHM PARAMETER

After we created the first package of digital musical effects for the DirectX environment – the Simple Audio Plug-In Pack (parametric equaliser, universal modulation effect and double delay) - we found that operating these universal effects was too complicated for some users. That is why we focused on the reduction and linking of algorithm parameters. As an example we will show the *Distortion* designed for electric guitars. It is a simplified algorithm of the universal effect based on spectrum modification caused by signal passage through a system with non-linear transfer characteristic, whose block diagram is in Fig. 3. If we leave aside parameters of amp simulator, which is formed by several filters connected in the series and in parallel, this algorithm has a total of 10 parameters: drive, warmth, type and order of LPF, cut-off, resonance, gain and delay in feedback, level and coefficients of the approximation polynomial.

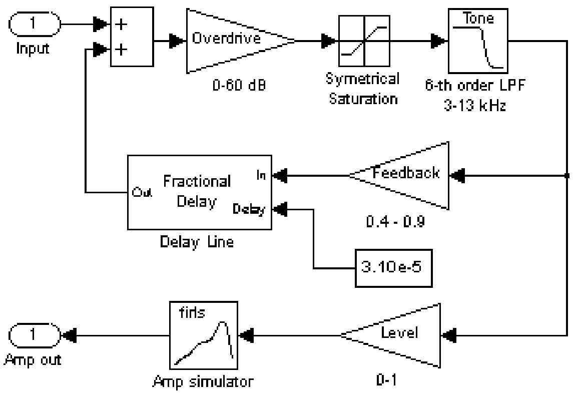

Figure 4: *Simulink model of guitar distortion effect*

As can be seen from Fig. 4, some blocks (and thus also parameters) were left-out, some parameters were experimentally set-up to constant values, and some parameters were linked with others. The resulting algorithm has 3 parameters (they are highlighted in Fig. 4):

- Overdrive – represents the distortion amount
- Tone – changes the sound colour in a certain range
- Level – sets the output level of effect (to prevent overloading the last DirectShow transform filter)

Parameter linking is denoted by means of dashed line in Fig. 4.

4. IMPLEMENTATION ON DIGITAL SIGNAL PROCESSORS

The fundamental element of most dynamic effects (e.g. tremolo, auto-wah, auto panner, etc.) is the low-frequency oscillator – LFO. The delay line is the basic element of delay effects (delay, echo, multi-tap echo, etc.). Modulation effects (vibrato, flanger, and chorus) use a combination of these two blocks – the delay line controlled by LFO. In this section we will deal with the implementation of these blocks on digital signal processors (DSP). The implementation of whole effects is already simple. We will not deal with the implementation of the parametric or the graphic equaliser because the linear filter DSP implementation problems have been described sufficiently, among others in [4] or [1]. General DSP features can be found, for example, in [1].

4.1. Low-frequency oscillator LFO

LFO parameters are frequency and the shape of generated signal. The frequency range is from tenths of Hertz to tens of Hertz. The sine, triangular, saw and rectangular waveforms are used most often. The triangular waveform is generated according to the equations

$$g(n) = g(n-1) + k \qquad \text{for the leading edge} \qquad (1a)$$

$$g(n) = g(n-1) + (-k) \qquad \text{for the trailing edge} \qquad (1b)$$

where *g(n)* is the triangular signal sample in *n*th step and *k* is the increment

$$k = \frac{4}{f_s} \cdot f_{LFO}\,, \qquad (2)$$

where f_s [Hz] is the sampling frequency and f_{LFO} [Hz] is the LFO frequency. The saw waveform is generated in the same way as the leading edge of the triangular waveform (1a). Since the duration of the leading edge of the saw waveform is twice than of the triangular waveform, the increment for the saw waveform must be half the increment in (2). The rectangular waveform is generated according to the triangular waveform – the value changes with the processor arithmetic overflow, i.e. with changing increment sign.

The sine waveform can be generated by various methods, e.g. by means of the Taylor series or we can use the triangular waveform. However, the optimal solution is applying the recursive generator in Fig. 5.

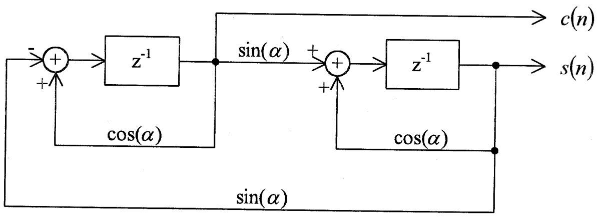

Fig. 5: *Recursive generator of sine and cosine waveforms*

The realisation equations of the recursive generator from Fig. 5 are

$$c(n+1) = c(n).\cos(a) - s(n).\sin(a) \qquad (3a)$$

$$s(n+1) = c(n).\sin(a) + s(n).\cos(a) \qquad (3b)$$

where *c(n)* is the cosine signal and *s(n)* the sine signal, and

$$a = 2 \cdot p \cdot \frac{f_{LFO}}{f_S}\,, \qquad (4)$$

where f_s [Hz] is the sampling frequency and f_{LFO} [Hz] is the LFO frequency. Initial conditions are *c(0)* = 1 and *s(0)* = 0. The realisation equation of the tremolo effect can be

$$y(n) = Gain \cdot [x(n) + Depth \cdot x(n) \cdot g(n)], \qquad (5)$$

where *x(n)* is the input and *y(n)* is the output sample, *g(n)* is the LFO sample, *Depth* is the modulation depth, and *Gain* is the signal gain.

4.2. Delay Line with Variable Delay

The delay line with variable delay can be realised using the shift register with FIFO structure. Input samples of the signal enter the register and they are shifted towards the output at the sampling frequency. The sampling frequency is controlled by the LFO and the latency time τ is given by the relation

$$t(n) = Depth \cdot g(n) \qquad (6)$$

where *Depth* is the depth of modulation, or magnitude of wobbling, and *g(n)* is the LFO sample. Multiplying the delay time by the sampling frequency yields a real number *K*, which gives the delay in the number of samples.

$$K = |t(n) \cdot f_S|\,. \qquad (7)$$

Since *K* is a real number, we have to carry out the interpolation of two neighbouring samples to obtain output sample x_D

$$x_D = x_0 \cdot (1 - fract(K)) + x_1 \cdot fract(K) \qquad (8)$$

where *fract(K)* is the decimal part of *K* and x_0 and x_1 are neighbouring samples:

$$x_0 = x'(n - Delay \pm ceil(K)) \qquad (9a)$$

$$x_1 = x'(n - Delay \pm ceil(K) \pm 1) \qquad (9b)$$

where *Delay* is the adjusted mean magnitude of delay and *ceil(K)* is the integer part of *K*.

The realisation equations for one channel of the modulation effect with delay line with variable delay can be, for example,

$$x'(n) = x(n) + Feedback \cdot x'_D \qquad (10a)$$

$$y(n) = Dry \cdot x(n) + Wet \cdot x'_D \qquad (10b)$$

where *x(n)* is the input and *y(n)* is the output sample, *x'(n)* is a sample on the input and $x'_D(n)$ on the output of the delay line, *Dry* is the input signal gain, *Wet* is the output signal gain, and *Feedback* is the feedback gain.

5. EFFECT IMPLEMENTATION ON PC/MAC BY THE PLUG-IN METHOD

In the case of implementation of digital musical effects by the Plug-In method, the algorithm realization is simple thanks to libraries with mathematical functions and the STL (Standard Template Library) with abstract data types. A greater problem is the implementation of these algorithms in the DirectX environment. That is why in this section we will deal with this implementation, and not with algorithms.

An example of the implementation of universal modulation effect by the Plug-In method including program printout in the C++ language can be found in [2].

5.1. DirectX technology

Microsoft DirectX gives applications a rapid access to the hardware of both present-day and future computers. It provides a consistent interface between the hardware and the applications, reduces the complexity of installation and configuration, and allows maximum utilization of the hardware. Programmers can make use of the potentialities of hardware without knowing it in detail.

DirectX contains a broad spectrum of technologies that standardizes the application of multimedia potentialities of computers. DirectX is divided into three basic levels:

• Components	– applications
• DirectX Media	– application services
• Direct X Foundation	– system services

5.1.1. Components

This level forms the apex of DirectX hierarchy. It is a group of applications that make use of the potentialities of lower levels. Those belonging here are NetMeeting, VRML, ActiveMovie and NetShow.

5.1.2. DirectX Media

This level contains the so-called "high-level API" access to the hardware. DirectX Media contains five APIs:

- Direct3D Retained Mode – services enabling the creation and animation of 3D objects
- DirectPlay – services creating standard multi-user interconnection of applications via Internet, a modem or network.
- DirectShow – universal architecture enabling the creation, processing and playback a multimedia flow of data. It is a initial technology used to create digital musical effects by the Plug-In method.
- DirectAnimation – enables adding media-, animation-, and interactive effects to Web sites and to multimedia applications.
- DirectModel – is a 3D graphic tool designed to solve the problem of large model of interaction.

5.1.3. DirectX Foundation

This is a heart of DirectX. It contains the so-called "low-level API" access to hardware that provides huge possibilities of hardware acceleration and also enables eliminating problems with the compatibility of hardware and system control elements.

It contains these basic elements:

- DirectDraw – provides direct access to graphic memory and is able to utilize the hardware properties of graphic cards
- Direct3D Immediate Mode – enables applying the properties of 3D hardware graphic cards and MMX technology
- DirectInput – provides direct connection with control elements (keyboard, mouse, joystick, ...)
- DirectSound – enables playback, recording, mixing, setting the volume level
- DirectSound 3D – enables 3D placing of sound, either by software or by hardware.

5.2. COM – Component Object Model

Most APIs in DirectX SDK consist of objects and interfaces based on the Component Object Model (COM). It Will be good to say a few words about this model.

The basic ideas of COM can be made clear on the basis of discussing the following problem:

1. Let there be an application (client) that needs to make use of discussing the function library, or
2. One application wants to make use of the service of another application that runs in a separate process, or
3. An application needs to make use of a service that runs on another computer.

A basic need of all these examples is making use of the services of another program. The mechanism of obtaining these services is different for each example. The solution lies in the definition of communication protocol, which is different for each example.

This problem is solved by the software architecture of COM, which defines the so-called objects and mechanisms, how they are constructed, destructed and mainly how they communicate with each other.

The COM architecture has these basic properties:

- It defines the binary standard
- It is programmer- and language-neutral
- It operates on many system platforms
- It provides a robust evolution in component-based applications and systems
- It uses a uniform model of communication between components in an application, between applications in a computer, and between computers

- It enables the components to share memory management
- It provides a rich error and state warning
- It enables a dynamic utilisation of component.

Since COM is a binary standard, it is language neutral. Any programming language can be used that enables creating tables of pointers to functions (C, C++, Smalltalk, Ada, and Basic) for the creation of a COM component.

5.3. DirectShow SDK

DirectShow SDK enables software developers to have access to *DirectShow* services that provide playback of the flow of multimedia data from local files or Internet servers, and importing multimedia from devices (graphic and sound cards).

5.3.1. *Architecture*

The hearth of *DirectShow* services is a modular system of components called “filter”. It should be noted that here the concept “filter” is not taken in the sense as we know it when processing signals (filtration) but as a key component of *DirectShow* architecture. An idea of this architecture can be obtained from the Figure of basic model [1].

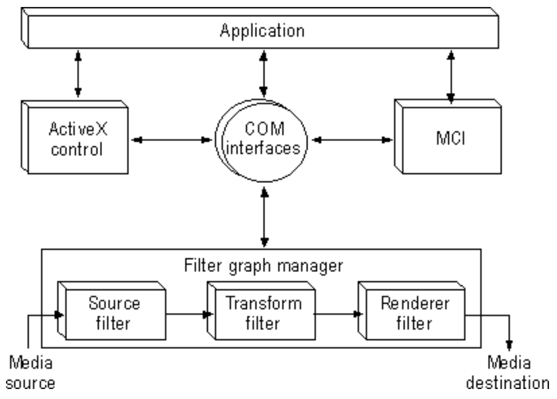

Figure 6: *DirectShow architecture*

The filters are arranged and configured in a component called “filter graph manager”. The filter graph manager provides filter interconnection, controls and checks the flow of data.

The application controls the activity of filters via communication with the filter graph manager, either directly by calling the methods of COM interface, or indirectly using the ActiveMovie ActiveX element or the so-called Media Control Interface (MCI).

5.3.2. *Filters*

The filter graph is made up of different filters. Most filters can be classed as belonging to one of three types:

- Source filter – a filter that provides flow od data from different souces
- Transform filter – a filter that performs data processing
- Rendering filter – a filter that transmits data to an output device

The filter graph can be created either by the programmer or automatically via teh filter graph manager and filter mapper.

The filter mapper tries to interconnect filters, from the source filter of a given type to the rendering filter.

5.3.3. *Filters and pins*

Two basic components are used when creating the filter graph: COM object filter and COM object pin.

Two types of pin are distinguished:

- Input pin – which admits data into the filter
- Output pin – which transmits data from the filter

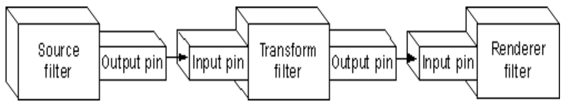

Fig. 7: *Filters and pins*

Pins ate responsible for creating a COM interface that enables connection with other pins and transport of data (IPin). Likewise, filters also have their interfaces (IBaseFilter).

5.3.4. *Data transport in filter Graph*

Data transport in the filter graph takes place in keeping with the following instructions:

- Media sample protocol – it defines the manner of data storage sharing and data exchange between filters
- End-of-stream protocol – it defines how filters generate and process information on data end
- Flushing protocol – it defines the flow of data via the filter graph
- Error detection and reporting protocol – defines how error and state reports are generated and processed
- Quality management protocol – enables dynamic adaptation to hardware and network conditions in order to improve quality

5.3.5. *C++ Classes*

DirectShow SDK contains C++ classes that help to create the required interface. The majority of base classes are directly related to the interfaces while some classes enable integration with the Win32 structure.

The most basic class creating an object is the *CBaseObject* class. All other classes are inherited from the *CUnknown* class, which creates the *INonDelegatingUnknown* interface, which, the same as the *IUnknown* interface in classical COM objects, takes care of a correct object aggregation.

The base classes create the following interface:

- Filter Base Classes – classes for creating filters
- Pin Base Classes – classes for pin operation

- Enumerator Base Classes – are used for the interfaces that enable an enumeration of filters and their properties.
- Transport Base Classes – realise the methods for memory management and data transfer
- Media Control and Positioning Classes – realise starting, stopping and changing the position
- Clock Base Classes – help to create the master clock in the filter graph

In addition to the base classes there are several useful classes here:

- Win32 Classes – these classes process the system events
- List and Queue Classes
- Multimedia Classes
- COM Classes – create a COM base interface
- Debugging Classes – facilitate filter development

5.4. Creating a Transform Filter

Creating digital musical effects by the Plug-In method is in fact creating a transform filter by means of DirectShow SDK. The same as a musical effect, a transform filter too, processes data on the input in a certain way and then sends these modified data to the output.

The following simplified steps show the basic procedure of creating the filter:

1. For the realisation to be as simple as possible, it is necessary to choose, depending on the required effect function, a suitable class, from which our effect will inherit its properties. This choice mostly depends on whether or not the filter has to copy input data. Accordingly, we choose either the *CTransform* or the *CTransformInPlace* class.
2. We must realise the *IUnknown* interface of our object.
3. We must define the class constructor and if we inherit from the *CTransformInPlace* filter we must realise the following methods:
    - *Transform* – which realises the effect proper
    - *CheckInputType* – which checks the type of input data
4. It is necessary to implement the *CreateInstance* method, which creates the filter object
5. It is necessary to define the self-registrerable *ClassFactory* template object
6. It is necessary to create GUIDs for our object.

6. CONTROL OF ALGORITHM PARAMETERS

For the control of parameters by means of hardware equipment we can use an arbitrary bus that is integrated on DSP or supported by personal computer (PC). In both cases the simplest thing is to use a serial bus. The communication rate of the serial bus is sufficient for the requirements of real-time control of effect parameters. An advantage is the hardware support by processor (interrupt) and simple operation.

In order to control musical effect parameters we can design our own communication protocol (see [4]) or we can implement on PC and DSP the MIDI interface, which is designed for these purposes.

The structure of the MIDI communication protocol allows controlling many parameters independently and setting the values of variables. It also enables both handshake and non-handshake transfers of large data blocks. In comparison with similar protocols an advantage of the MIDI is not only its suitable structure but also easy availability of low-cost hardware for recording, editing, and playing MIDI data, and of control units (so-called controllers) of various designs, including breath and the pressure controllers.

In [6] you can find a description of the MIDI interface as well as a MIDI programming model. We will deal with the MIDI implementation on PC and DSP in detail in our partial research report on the solution of project No OC G6 10 for the year 2000.

7. CONCLUSIONS

Two packages of digital audio effects – Simple Audio Plug-In Pack and Stomp’n FX vol. 1 – for PC/Mac platform are the results of our work. All effects are implemented for the DirectX and VST (ASIO, ASIO2) environments. At present we are finalizing another package of effects –Stomp’n FX vol.2 and we are preparing the implementation of these effects on DSP Motorola, MIDI interface support, algorithms for the detection of BPM of music signal, etc. Information about our products can be found at http://www.dsound1.com. The results of our work will be available in our partial research reports on the solution of project No OC G6 10.

8. REFERENCES

[1] Smekal, Z. et al., Partial research report on the solution of international project No OC G6 10 for the year 1999, 1999.

[2] Smekal, Z. et al., Partial research report on the solution of international project No OC G6 10 for the year 1998, 1998.

[3] Balik, M., Oboril, D., “Digital Musical Effects by the Plug-In Method”, Competitive review of Student Member Scholarly Works at the 108th International Convention of The Audio Engineering Society, Student Delegate Assembly, 2000.

[4] Dobes, F., “Digital Audio Effects on the Motorola DSP56002 Digital Signal Processor”, Diploma Thesis, Department of Telecommunications FEECS, Brno University of Technology, 2000.

[5] Schimmel, J., “Digital simulation of effects based on transfer characteristic non-linearity”, Proceedings of International Scientific Conference TSP, p. 449-451, 2000.

[6] Schimmel, J., “Musical Instruments Digital Interface for the Motorola Digital Signal Processor”, Proceedings of International Conference Research in Telecommunication Technology, p. 85-88, 2000.

[7] Benson, K.B., Audio Engineering Handbook, Mc Graw-Hill, New York, 1988.