Buletinul Stiintific al Universitatii "Politehnica" din Timisoara, ROMANIA
Seria AUTOMATICA si CALCULATOARE (ISSN 1224-600X)
Transactions on AUTOMATIC CONTROL and COMPUTER SCIENCE, Vol. 46(60), 2001

# Implementing Professional Audio Effects with DSPs

Mihai V. Micea [1], Mircea Stratulat [2], Dan Ardelean [3]
and Daniel Aioanei [4]

Software and Computer Engineering Department, POLITEHNICA University of Timisoara,
2, V. Parvan Blvd, 1900 – Timisoara, Romania, Phone/Fax: +40 56 192049

[1] E-Mail: micha@dsplabs.utt.ro, WWW: http://dsplabs.utt.ro/~micha
[2] E-Mail: smircea@cs.utt.ro, WWW: http://www.cs.utt.ro/~smircea
[3] E-Mail: dardelean@dsplabs.utt.ro, WWW: http://dsplabs.utt.ro/~dardelean
[4] E-Mail: daioanei@dsplabs.utt.ro, WWW: http://dsplabs.utt.ro/~daioanei

***Abstract** – Digital signal processors are highly used in a large variety of today's domestic and industrial applications. This paper emphasizes on the principles and techniques involved in professional audio processing field. Some basic digital audio effects are described along with the corresponding guidelines for DSP implementation.*

***Keywords:** DSP, digital audio effect, chain of effects, real-time, audio buffer, communication protocol*

## 1. INTRODUCTION

Professional audio processing has developed rapidly in the last years, from predominant analog techniques to digital processing algorithms and systems.

As result, a digital system to implement a professional audio processor needs to meet high-performance requirements: high-quality processing of multiple audio channels, real-time operation capability, stability, reliability and scalability [4].

On the other hand, as new audio effects are continuously developed, the audio processor must be able to implement them with little effort from the user/developer.

A convenient solution to this problem is to design the audio processor to provide plug-in capabilities, with the audio effects implemented as run-time loadable modules.

In signal-based applications involving digital processing, the most efficient implementation solution is using specialized family of microprocessors: the digital signal processors (DSPs).

DSPs feature high performance, autonomy and flexibility at effective costs. Their internal architecture is particularly designed to execute signal-processing instructions in parallel, using specific hardware structures (like Harvard bus structure, register file, circular memory buffers and special internal memory and ports).

In addition, many digital signal processors, like the Motorola DSP56037, feature a highly specialized internal coprocessor (EFCOP – Enhanced Filter COProcessor), able to execute autonomously various filtering and time-consuming data manipulation algorithms.

In this paper we focus on the basic issues involved in the design and implementation of a professional audio processor with digital signal processors.

The next section describes some basic audio effects, their general behavior and characteristics as algorithms designed for digital system implementation, considering as target a generic DSP.

Further on, the paper discusses the problem of managing and interfacing the audio effects to the user/developer.

As conclusions, we describe our practical approach of this subject – a digital audio effects processor implemented with the Motorola DSP56307, along with its general characteristics and some performance considerations.

## 2. AUDIO EFFECTS IMPLEMENTATION

Most of the audio effects used in professional sound editing and processing application can be designed in terms of signal processing algorithms, suitable for implementation on digital systems, particularly on digital signal processors.

Some basic digital audio effects are described in the following paragraphs, along with the corresponding guidelines for DSP implementation.

### *2.1. Volume Effect*

In its simplest form, the volume effect controls the amplitude of the signal by varying the attenuation of the input signal [3].

However, an active volume control will have the ability to increase the volume (i.e. amplify the input signal) as well as attenuating the signal:

$$y[n] = \upsilon \cdot x[n] \tag{1}$$

where:

$y[n]$ is the current sample of the output signal,

$x[n]$ is the current sample of the input signal,
$\upsilon$ is the volume scaling factor:
$\upsilon = 1$, results in no effect on the output signal;
$\upsilon < 1$, the signal amplitude is decreased;
$\upsilon > 1$, the signal amplitude is increased.

For DSP implementation, the current audio sample can be stored in an internal register and the volume factor as a memory variable:

```
sample[i]= sample[i]·vol;
```

Volume controls are useful for placing between effects, so that the relative volumes of the different effects can be kept at a constant level.

### *2.2. Panning*

Panning is used in stereo recordings. The volume of each channel can be adjusted. This effectively adjusts the position of the perceived sound within the stereo field.

The two extremes are: all sound completely on the left or all sound completely on the right. This is commonly referred to as balance on commercial sound systems [3, 4].

The effect implies two parameters as two decays – one for the left channel and one for the right channel (`vol_left` and `vol_right`).

Varying synchronously the two parameters, the panning effect is achieved. The two parameters can appear as one synchronized parameter to the user.

On the DSP, the signal transforming would appear as follows:

```
output_left = input_left·vol_left;
output_right = input_right·vol_right;
```

### *2.3. Chorus*

The chorus effect is achieved by adding an echo to the original signal and then varying the delay of the echo between a maximum and a minimum delay value at a certain rate [3].

The algorithm describing the chorus effect is as follows:

$$y[n] = x[n] + x[n - \Delta[n]] \tag{2}$$

where:

$\Delta[n]$ : $\mathbf{Z} \rightarrow \mathbf{Z}$, is a discrete and periodic delay function. It can be, for example, a triangular function (see Fig. 1 and Fig. 2).

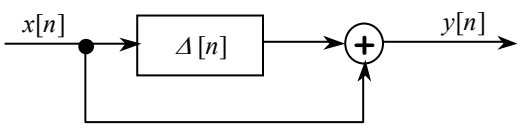


Fig. 1. Chorus and Flanger effect diagram


Fig. 2. Two common waveforms for delay variation (source: [3])

The delay increases until it reaches a certain value and then decreases to a minimum value.

The maximum value and the rate of raising and falling of $\Delta[n]$ function can be treated as parameters for this effect. Typically, the delay of the echo for this effect varies between 40 msec and 60 msec at a rate of about 0.25 Hz.

On the DSP, chorus can be implemented using a circular buffer for audio sample processing. The output sample results as the sum of the current input sample and a delayed sample:

```
buffer[i] = input[i];
output[i] = buffer[i]+ buffer[i-offset[i]];
```

The delayed sample will be retrieved from the buffer at a variable offset, calculated with the $\Delta[n]$ function, between a minimum value (e.g., 1) and the maximum value set currently by the user.

### *2.4. Flanger*

Flanging is a special case of the chorus effect: it is created in the same way that chorus is created, that is mixing a signal with a delayed copy of itself [4].

Typically, the delay of the echo for a flanger is varied between 0 msec and 10 msec at a rate of 0.5 Hz (see Fig. 1).

The delay of the signal cannot be heard by a human ear, but it creates a series of notches in the signal. The delay can be varied following either a sine wave or a triangle, as shown in Fig. 2. The most common in flangers is the triangle.

The ecuation for this effect is similar to (2).

### *2.5. Phaser Effect*

Phasing is very similar to flanging. If two signals that are identical, but out of phase, are added together, then the result is that they will cancel each other out.

If, however, they are partially out of phase, then partial cancellations and partial enhancements occur. This leads to the phasing effect [3, 4].

In the algorithmic form, the phaser effect is given by the equation below:

$$y[n] = x[n] - \gamma \cdot x[n - d] \tag{3}$$

where:

$\gamma$ is a proper scaling factor (decay),
$d$ is a discrete delay.

The implementation of the phaser effect on the DSP involves a circular audio buffer, similar to the one described at the chorus effect.

In algorithmic form, the phaser effect performs the following steps:

```
buffer[i] = input[i];
output[i] = buffer[i] –
          - decay · buffer[i - delay];
```

A diagram of this effect is shown in the figure bellow (APF meaning "All-Pass Filter"):

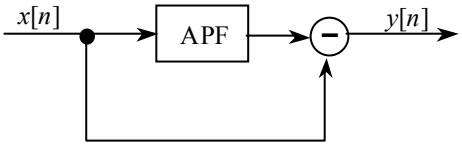


Fig. 3. Block diagram of the phaser effect

*2.6. Reverb Effect*

Reverb is used to simulate the acoustical effect of rooms and enclosed buildings. In a room, for instance, sound is reflected off the walls, the ceiling and the floor. The sound heard at any given time is the sum of the sound from the source, as well as the reflected sound.

An impulse (such as a hand clap) will decay exponentially. Concert halls and rooms have to be designed such that the reverberation effect is adequate for the type of sound that will be produced.

The digital reverberator is implemented by using a number of comb filters ("CF") in parallel with varying delay times. The outputs of the comb filters are summed together through an all-pass filter ("APF") to produce the reverb effect [3, 4]. The following diagram shows those explained:

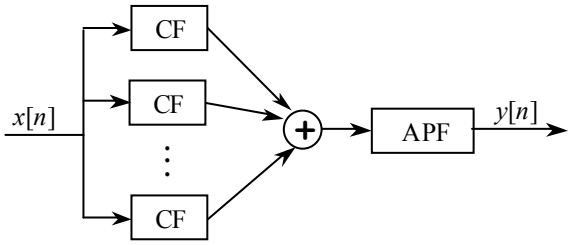


Fig. 4. Digital reverberator diagram

The implementation of the reverb effect on the DSP includes several comb filters. The reverb effect can share the same buffers that the echo effect is using.

But, unlike the echo algorithm, the reverb does not store the modified sample (the resulted sample) in the buffer. Instead, it stores the original sample.

The general description of the reverb effect as a digital system is presented following algorithm:

$$y[n] = x[n] + \gamma \cdot x[n-d_1] + \gamma \cdot x[n-d_2] + \\ + \gamma \cdot x[n-d_3] + \gamma \cdot x[n-d_4] + \\ + \gamma \cdot x[n-d_5] + \gamma \cdot x[n-d_6] \quad (4)$$

The delays of the comb filters ($d_i$ in the previous equation) and the general scaling factor (decay, $\gamma$) can be viewed as the reverb effect main parameters.

*2.7. Echo*

A basic echo effect can be obtained by simply adding a sound sample from the past to the current sound sample.

Therefore the simple echo effect involves two basic operations: time-delay and addition, to generate the output sample from the current one and the delayed sample (see Fig. 5). The delay time can vary from a few miliseconds to a few seconds.

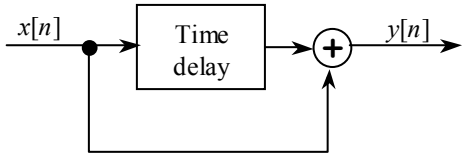


Fig. 5. Diagram of an echo device

This device however, produces a very simple kind of echo [3, 4]. A better echo device uses multiple echoes, adding a feedback control which takes the output of the delay block and takes it back to the input through an attenuator ("FA"), as shown in Fig. 6.

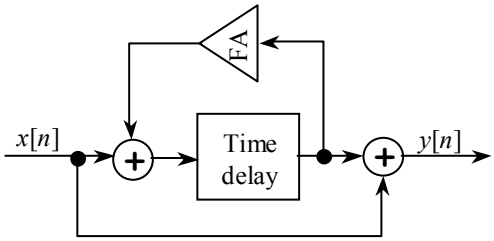


Fig. 6. Diagram of an echo device with feedback

The attenuator determines the decay of the echoes (feedback gain should be less than 1), which is how quickly each echo dies out.

The echo effect is described by a first order difference equation since it involves a simple feedback loop:

$$y[n] = x[n] + \gamma \cdot y[n-d] \quad (5)$$

where:

$y[n - d]$ is the sound sample from the past with its corresponding gain, which actually creates the echo,

$d$ is the user-adjustable delay (i.e., the actual delay after which the echo will be heard),

$\gamma < 1$ is the feedback gain. It controls in exponential manner how long each echoed sample can be heard.

On the DSP, echo can be implemented using a circular buffer for each audio channel to be processed. The buffer contains the echo history at any time (i.e., the $y[n-1]$ term from (5)).

At each moment, the current audio sample is added to a value from the buffer and the result is both sent to the output and stored back into the buffer.

Considering the echo buffer length of $N$ locations, the basic algorithm for the echo effect involves the following steps:

```
output = input + gain·buffer[k - delay];
buffer[k] = output;
k = (k+1) mod N;
k - delay = (k - delay) mod N;
```

where:

$k$ is a pointer to the current location in the buffer,

*gain* is the feedback attenuation factor that can be controlled by the user ($gain < 1$).

### 2.8. Ring Modulation

This effect takes two signals and multiplies them, thus producing a signal that contains the sums and differences of the frequencies of the two original signals. These frequencies will typically be non-harmonic, so the ring modulator can yield some very dissonant sounds. For that reason ring modulator is not widely used [4].

By multipying the two input signals, the amplitude modulation is implemented (more specifically, suppressed-carrier modulation):

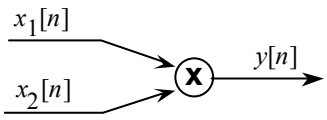


Fig. 7. A ring modulator multiplies two input signals

$$y[n] = x_1[n] \cdot x_2[n] \qquad (6)$$

The ring modulator can have only one input connected to an instrument and the other input to an internal oscillator (which is generally considered to be the carrier signal). Since the carrier signal doesn't appear in the output, it is called "suppressed carrier".

Fig. 8 exemplifies the effects of a ring modulator.

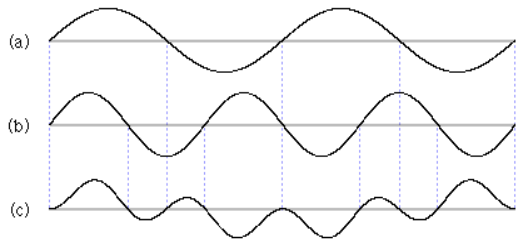


Fig. 8. (a) 200 Hz sine wave, (b) 300 Hz sine wave, and (c) the product of the two. The product is zero when either wave is zero (source: [3]).

### 2.9. Equalizer

Equalization is an effect that allows the user to control the frequency response of the output signal. The user can boost or cut certain frequency bands to change the output sound as needed.

It is usually performed with a number of bandpass filters, all centered at different frequencies (outside each other's frequency band), and the bandpass filters have controllable gain [5].

As the human ear perceives sounds in a logarithmic manner, the center frequencies, as well as the pass-bands of the filters must be calculated on a logarithmic scale (see Fig. 9).

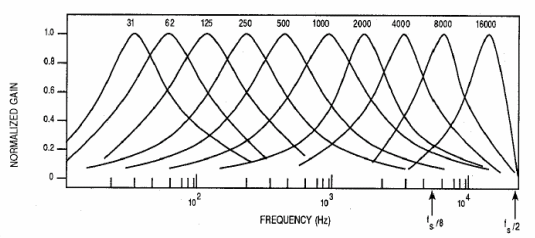


Fig. 9. 10-band bandpass IIR filter response (source: [6])

The 10-band stereo equalizer is implemented using 10 digital IIR bandpass filters in parallel for each stereo audio channel.

Assuming the audio codec samples the incoming audio stream at a rate of $f_S$ Hz, the center frequencies for these filters lie between 0 Hz to $f_S/2$.

At each sample period, a left and right sound sample is fed into the 10 digital filters. After each individual bandpass filter eliminates the frequencies not in its range, the output is scaled by a corresponding gain. Finally, the results of the ten filters are summed together.

This process allows one to selectively remove or limit, the gain of a particular frequency range from the sound source (see Fig. 10).

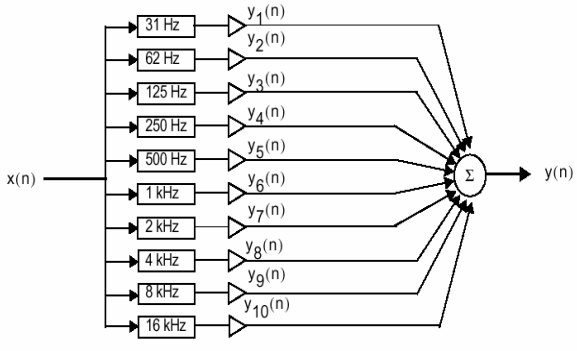


Fig. 10. IIR equalizer data-flow diagram (source: [6])

Each bandpass filter is designed in a similar manner, following the analog passive "RCL Bandpass Network", with identical quality factors and with the central frequencies based on equal intervals of the log of frequency [5].

Hence, the IIR difference equations of the bandpass filters follow an identical pattern:

$$y[n] = 2 \cdot (\alpha(x[n] - x[n-2]) + \gamma y[n-1] - \beta y[n-2]) \quad (7)$$

where:

$\alpha$, $\beta$ and $\gamma$ are the filter coefficients (see Fig.11).

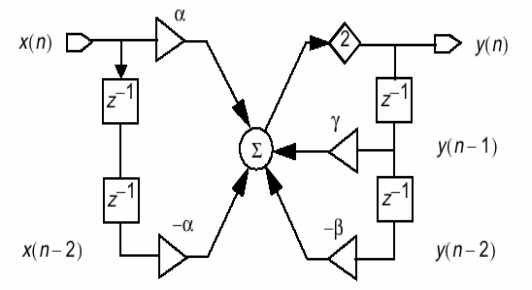


Fig. 11. Bandpass IIR Filter Network Diagram (source: [6])

The bandpass filter coefficients in (7) can be calculated for each filter knowing its particular center frequency, $\theta_C$ and the filter quality factor, $Q$ (same for all filters):

$$\alpha = \frac{\frac{1}{2} - \beta}{2}; \quad \gamma = \left(\frac{1}{2} + \beta\right)\cos\theta_C \quad (8)$$

and

$$\beta = \frac{Q - \frac{\theta_C}{2}}{2Q + \theta_C} \quad (9)$$

## 3. INTERFACING AND MANAGING THE AUDIO EFFECTS PROCESSOR

Interfacing the audio effects and controlling their parameters can be made using different approaches.

This paper emphasizes on a simple and efficient solution: a special-purpose communication protocol and a proper framework for managing and controlling the audio effects.

The communication protocol is designed to provide a real-time data link between the DSP – running the audio processor, and a host computer – running the user interface.

To provide easy control of the system parameters and to facilitate integration of new audio effects, we propose a modular approach. All the audio effects are implemented as stand-alone modules that can be activated by the user into a "chain of effects" to be run on the DSP.

### *3.1. Communication protocol*

The communication protocol establishes a link between the DSP and a generic control and monitoring application, API (this could be a keyboard, a PC application or any other device capable of low-level serial protocol).

The protocol is based on control words, each word having a length of 3 bytes.

The first byte specifies the parameter number to be read/changed by the monitoring and control application and the last two bytes specify a 16 bit value.

Protocol tasks:

(a) Read/change parameter value
(b) Synchronization
(c) Real-time communication

(a) Read/change parameter value

After each parameter ID (first command byte) received from the API, the DSP module sends an echo value, signaling that it has correctly received the parameter ID.

Read is performed by specifying the parameter ID in the first command byte and the hex value of `$FFFF` as the parameter value. This prevents the DSP module from altering the parameter value. It also sends back the current value of the parameter specified by its ID.

Change is performed by specifying the parameter value which must be different from the hex value of `$FFFF`. The DSP module answers with the old value of the parameter, acknowledging that data was correctly received.

(b) Synchronization

A special parameter ID (`$FF`) is reserved for synchronization. The value stored by the DSP module for this parameter is `$55AA`.

To synchronize with the DSP module, the API has to send a sequence of bytes with the hex value of `$FF` until it receives a special sequence from the DSP module: "`$FF $55 $AA`". This way the API will not to change any value of the parameters held by the DSP module.

After receiving the "`$FF $55 $AA`" sequence, the API is synchronized with the DSP module and is able to query or change parameters (the communication protocol treats the active chain of effects as a set of parameter IDs).

(c) Real-time communication

The DSP module performs a real time loop: receive the audio signal sample, process it and send the processed sample to the output.

To ensure the real time characteristics of the audio process, the DSP module must not wait for the whole communication word to be received.

After each sample is processed, the communication channel is checked for a new constructed byte. Then the byte is stored into a special buffer and when the full three-byte word is ready, the corresponding parameter is eventually changed.

### *3.2. Managing Audio Effects*

The following diagram shows the actions taken by the DSP module to apply the audio effects to the input digital signal:

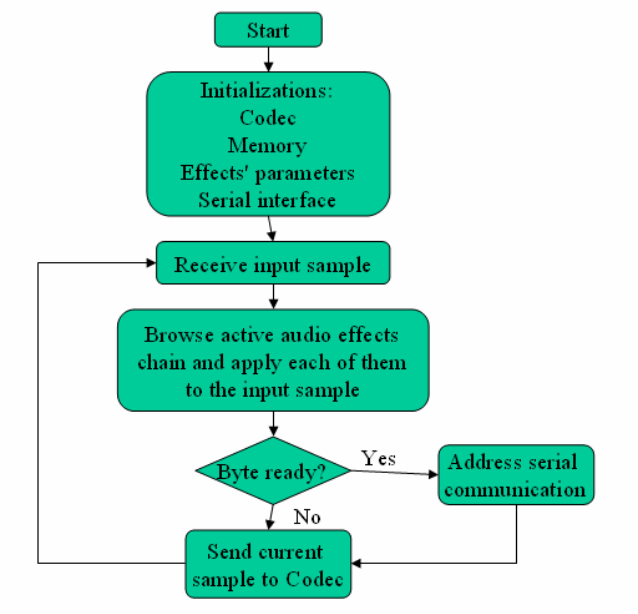


Fig. 12. Audio effects management diagram

This loop should be optimized in order to consume as little time as possible, thus leaving time for the audio effects to process the input digital signal.

Therefore, considering the audio signals processed at a sampling rate of $f_S$ , the overall time available to process one audio sample through the entire chain of effects is:

$$\tau_{sample} = \frac{1}{f_S} \quad [\mu \text{sec}] \qquad (10)$$

With $\tau_{sample}$ and the DSP clock rate, the maximum number of instructions allowed for the chain processing can be calculated.

In order to do this the loop has to take advantage of the DSP parallel architecture. For easier development, the architecture should be modular, and the audio effects should be of plug-in type.

## 4. CONCLUSIONS

This paper discusses the basic issues involved in designing a professional audio effects processor as a digital system, presenting the corresponding algorithms of some audio effects.

It also provides the main guidelines to implement the algorithms with a generic digital signal processor.

As an application, we developed an audio effects processor with the Motorola DSP56307 digital signal processor.

Motorola DSP56307 is a 24-bit processor, working at a maximum rate of 100 MHz, thus providing 100 MIPS of processing power.

It features on-chip RAM memory of 64 Kwords, a highly parallel instruction set, EFCOP (Enhanced Filter COProcessor) running concurrently with the core, internal timers and peripheral expansion ports, and effective off-chip memory expansion capabilities [2].

A 16-bit CD-quality audio codec is used for interfacing the sound channels with the audio processor. The codec operates at 48 KHz sampling rate and is directly connected to the DSP [1, 2].

The audio effects processor consists of two main modules: the DSP and the PC modules, communicating through a special-purpose serial link.

The DSP module implements both the active chain of audio effects and the serial communication protocol with the host computer. It is designed as a modular, open architecture, with the audio effects developed as distinct, plug-in modules.

Currently, the DSP module can process in real-time a total of nine audio effects. The DSP clock rate is set to 88.47 MHz and the total time available for processing one sample of sound through the entire chain of audio effects is $\tau_{sample}$ = 20.8 μsec.

Therefore, to maintain the real-time behavior of the audio processor, there are up to 1840 clock cycles available to process the sound sample.

The PC module implements the graphical user interface of the audio processor, as well as the serial communication link with the DSP module.

It is able to synchronize with the DSP module while the user can read or modify both the chain of active audio effects, as well as their parameters.

Our approach demonstrates the effectiveness of using highly-specialized processors – alias DSPs – to implement professional applications in today's increasing audio demands.

## ACKNOWLEDGMENT

This paper is part of the research and development program carried out by DSPLabs in close cooperation with the Motorola Corporation, USA.

The authors are members of the DALT (Motorola DSP Application Lab in Timisoara) project team, involved in developing the application grant DALT.2.6/2000 – "Audio Effects Processor with the Motorola DSP56300 Family".

## REFERENCES

[1] Motorola, Incorporated, "DSP56300 Family User's Manual", 1999.
[2] Motorola, Incorporated, "DSP56307 24-Bit Digital Signal Processor User's Manual", 1999.
[3] Scott Lehman, "Effects Explained", Harmony Central, (http://www.harmonycentral.com), 1996.
[4] Toby Kurien, "Audio Effects", (http://users.iafrica.com/k/ku/kurient/dsp/effects.html), 1997.
[5] James M. Montgomery, "Implementing a 10-Band Stereo Equalizer on the DSP56311EVM Board", Motorola Application Note, AN2110/D, 2001.
[6] Motorola, Incorporated, "Digital Stereo 10-Band Graphic Equalizer Using the DSP56001", Motorola Application Note, APR2, 1988.