OPTIMIZING DIGITAL MUSICAL EFFECT IMPLEMENTATION FOR HARVARD DSP ARCHITECTURE

Zdenek Smekal, Jiri Schimmel, and Petr Krkavec

Department of Telecommunications
Faculty of Electrical Engineering and Computer Science
Brno University of Technology
smekal@utko.fee.vutbr.cz, schimmel@utko.fee.vutbr.cz

ABSTRACT

In the area of digital musical effect implementation, attention has lately been focused on computer workstations designed for digital processing of sound (DAW – Digital Audio Workstation), which perform all operations with audio signals, such as routing, mixing, editing, effect processing, recording, coding, etc. They are in fact a combination of a powerful computer program and hardware cards with digital signal processors. Most of these operations, except editing, must be performed in real time. Until recently, all real-time operations were performed on digital signal processors. Thanks to the power enhancement of personal computer core, performing these operations in the CPU is currently possible. However, in most cases, digital signal processors are still used for these purposes because digital musical effect modelling is more effective and more precise with the digital signal processor. In addition to this, processing in digital signal processor saves the CPU computing power for other functions. This paper deals with optimizing algorithms of digital musical effects for DAW systems.

1. DIGITAL MUSICAL EFFECTS IN DAW SYSTEMS

We can split the software-produced digital musical effects that are used in DAW systems, into two groups – effects called DSP effects, which run in digital signal processors on hardware cards of these systems, and effects processed directly in the processor of a personal computer, which are called CPU effects. Sometime the former type of processing is referred to as Native Signal Processing and the other type as Host-based signal processing.
There is no difference between the DSP and CPU usage from the user's point of view. From the technical angle, however, the algorithm of the former type of effect is transferred in the signal processor assembler into the memory of signal processor placed on the PCI card and the computer is no longer concerned with it. In the other case the process that realizes the selected musical effect algorithm is started directly in the host computer – it means that the next process (which absorbs computing power) is invoked.

1.1. Plug-In Technology

A plug-in is a virtual module that is inserted into the processing path of a digital audio signal, similar to the effect inserted into the signal path of a mixing desk channel. The idea of plug-in technology is as follows: we create a hardware-independent program module that processes the signal with an exactly defined format of input and output. Then we create a host environment that will provide for this module links to this environment, i.e. it will offer standardized interfaces for communication with the hardware. See [1] for details.

1.2. Current DAW Systems

Current DAW systems like the ProTools by Digidesign and the Pulsar/Scope system by Creamware use their own DSP and CPU effect technology. They are very powerful systems but they have one disadvantage – for effects they use firm's own interfaces and thus they can only be used in a single program. However, systems that use the VST (Virtual Studio Technology) interface by Steinberg [2] for DSP effects appeared recently on the market. The TC Works and Universal Audio companies developed TC Powercore and UAD1 DSP cards that support DSP effects with the VST interface. Today, there are many programs on the PC and Mac platforms that support the VST interface. Thanks to the TC Powercore and UAD1 cards these programs will be able to use DSP effects too.

2. OPTIMIZING DIGITAL SIGNAL PROCESSING ALGORITHMS FOR DSP

Depending on the signal processor structure, digital signal processing algorithms for digital signal processors can be further optimized such that they reach maximum power in the DAW systems and thus match the plug-in architecture.
Many factors affect the precision of digital signal processing algorithms implemented on digital signal processors. With discrete linear systems it is mainly the setting of initial conditions. When we use a floating-point processor, the quantization of signal and, above all, the quantization of transfer function coefficients also influence the precision. The architecture of arithmetical-logic unit is also important – due to the limited ALU range there may appear saturation during computation and this can cause non-linear distortion or change in the frequency response of linear system. The Motorola DSP56k family digital signal processors [3] are used in most DAW systems so we dealt with implementation and optimization in this DSP family. We do not deal with digital audio effect algorithms themselves in this paper, but only with optimizing their

algorithms for Motorola DSP56k processors. A description of the effects mentioned in this paper can be found in [4].

2.1. Optimizing Parametric Filter Algorithms

Parametric filter structures allow direct access to the parameters of the transfer function such as gain, bandwidth and center or cut-off frequency, via the associated coefficients. To modify one of these parameters it is therefore no longer necessary to compute a complete set of transfer function coefficients but only one coefficient in the filter structure is calculated instead. A feed-forward structure for boost and feed-backward structure for cut achieve an independent control of gain, center/cut-off frequency and bandwidth. See [1] for details.

Any designed parametric filter is the linear time-invariant (LTI) discrete system of the s^{th} order. Such a system is usually described by LTI difference equation

$$\begin{aligned} &b_s y(n+s)+b_{s-1}y(n+s-1)+...+b_1 y(n+1)+b_0 y(n)= \\ &=a_s x(n+s)+...+a_2 x(n+2)+a_1 x(n+1)+a_0 x(n) \end{aligned}, \quad (1)$$

where $y(n)$ is the output signal, $x(n)$ is the input signal, and a_i , and b_j are the time independent coefficients. The initial conditions are defined by $2s$ values of $y(0), y(1), \dots , y(s-1)$ and $x(0), x(1), \dots , x(s-1)$. A more effective description of the discrete system is the state-space representation. Difference equation (1) can be rewritten as two state-space equations of the first order

$$\mathbf{v}(n+1)=\mathbf{A}\mathbf{v}(n)+\mathbf{B}x(n) \quad (2)$$

$$y(n)=\mathbf{C}\mathbf{v}(n)+\mathbf{D}x(n)\,. \quad (3)$$

The state-space vector is defined by state-space variables $\mathbf{v}(n)=[v_1(n),v_2(n),...,v_s(n),]^{\mathrm{T}}$ Matrices **A,B,C**, and **D** define the properties of the discrete system. There are only s initial values $v_1(0), v_2(0), \dots , v_s(0)$. Eq.(2) is made practical use of in the Matlab program. If $Y(z)$ and $X(z)$ are the z-transforms of $y(n)$ and $x(n)$, respectively, then the solution of Eqs.(2) and (3) can be shown as

$$Y(z)=H(z)X(z)+\sum_{i=1}^{s}H_i(z)v_i(0)\,. \quad (4)$$

The z-transform of the total response $Y(z)$ in Eq.4 is given by the sum of the ZSR (Zero State Response – the first term of Eq.4) and of the ZIR (Zero Input Response – the second term of Eq.4). Therefore, the transfer functions $H_i(z)$ constitute the transient response as a response to initial conditions. It is sometimes difficult to determine this response. A new method has been suggested that uses signal flow graphs to describe difference equations.

2.1.1. Generalized Models for Digital Signal Processor

The digital signal processor architecture is typically a simple or dual Harvard architecture. In order to reduce the number of iterations in the design it is desirable to be able to predict the performance of the digital filter hardware before actually implementing the filter. This includes the complexity of the structure, finite word-length effects, the prediction of arithmetic saturation, and design flexibility with respect to changing specifications.

There is a list of practical rules, which can be recommended to optimize the digital filter implementation on a DSP:

- Lower quantization effect sensitivities can be obtained by parallel or series connections of the first- and second-order partial sections. It is possible to use other structures (lattice or continued fraction expansion, etc.). The system transfer function $H(z)$ must be stable and may have real coefficients.
- The maximum values of the coefficients must lie inside two's complement range, i.e. inside the interval ±1. It is useful to divide by 2 all coefficients of the second-order partial section.
- For effective implementation, the coefficients of 2nd order partial sections b_2 are set to one. The optimized state-space difference equations of the second-order partial sections have the form

Canonic Model 1

$$\begin{aligned} &v_1(n+1)=a_1\,x(n)-b_1\,y(n)+v_2(n)\,, \\ &v_2(n+1)=a_0\,x(n)-b_0\,y(n)\,, \\ &y(n)=\frac{1}{b_2}(\,v_1(n)+a_2\,x(n)\,)\,. \end{aligned} \quad (5a)$$

Canonic Model 2

$$\begin{aligned} &v_1(n+1)=v_2(n)\,, \\ &v_2(n+1)=\frac{1}{b_2}(\,x(n)-b_1\,v_2(n)-b_0\,v_1(n)\,)\,, \\ &y(n)=a_2\,v_2(n+1)+a_1\,v_2(n)+a_0\,v_1(n)\,. \end{aligned} \quad (5b)$$

If Eqs (5) are solved by the z-transform, the signal flow graphs (SFGs) of the generalized models of the partial section can be obtained as shown in Fig.1 and Fig.2.

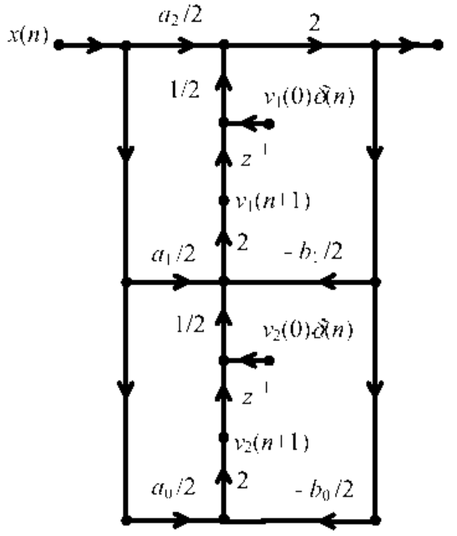


Figure 1. *SFG of the generalized canonic model 1 for DSP*

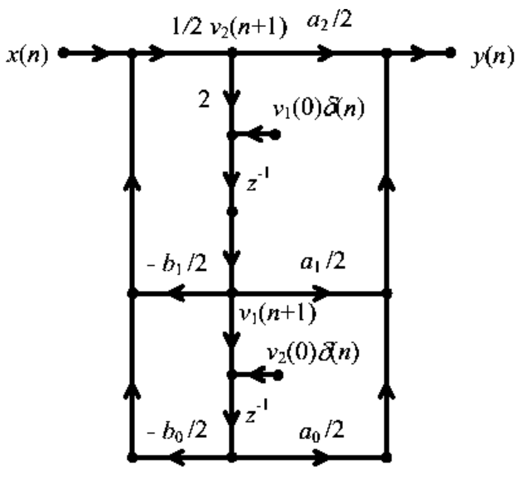


Figure 2. *SFG of the generalized canonic model 2 for DSP*

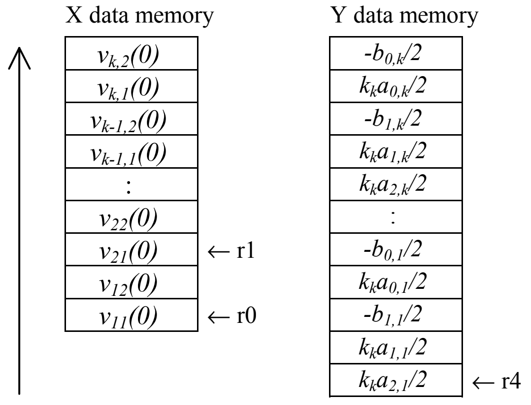
Both model 1 and model 2 have the same system transfer function *H(z)*, which can be calculated by Mason's Gain Rule as a path from input node *x(n)* to output node *y(n)*. The other transfer functions, $H_1(z)$ and $H_2(z)$, are also obtained by Mason's Gain Rule as the path from input nodes $v_1(0)\delta(n)$ and $v_2(0)\delta(n)$ respectively, to output node *y(n)*. To sum up, we can see that having the same *H(z)* both models have the same ZSR, but there are great differences in the ZSRs. The results are summarized in Table 1. We can expect much worse implementation properties for canonic model 2 than for canonic model 1 in spite of the fact that canonic model 2 is preferred in most world-known books! Definitely, the implementation of IIR digital filters on Motorola's fixed-point DSP56002 confirms these assumptions. Moreover, the behaviour of the state-space variables of the two models is different.

Table 1. Transfer functions from Eq.4 for partial section of the 2nd order ($b_2 = 1$)

	Model 1	Model 2
$H(z)$	$\frac{a_2z^2+a_1z+a_0}{z^2+b_1z+b_0}$	$\frac{a_2z^2+a_1z+a_0}{z^2+b_1z+b_0}$
$H_1(z)$	$\frac{z^2}{z^2+b_1z+b_0}$	$\frac{(a_0-b_0a_2)z^2+(a_0b_1-b_0a_1)z}{z^2+b_1z+b_0}$
$H_2(z)$	$\frac{z}{z^2+b_1z+b_0}$	$\frac{(a_1-b_1a_2)z^2+(a_0b_2-b_0a_2)z}{z^2+b_1z+b_0}$

The following example shows the core of optimized IIR filter algorithm using canonic model 1 for Motorola DSP56k family processors. Symbols $v_{k,2}$ and $v_{k,1}$ denote the state-space variables of kth section and symbols $a_{0,k}$, $a_{1,k}$, $a_{2,k}$, $b_{0,k}$, and $b_{1,k}$ denote the coefficients of kth section transfer function. K_k is the scale coefficient of kth section. Register *y1* contains an input sample at the beginning and output sample at the end of algorithm, *m* is the number of 2nd-order sections.

Memory usage:



```
      move    x:(R0),a
      do      #m,_end
      macr    y1,y0,a   x:(R1),b    y:(R4)+,y0
      asr     b         a,x0
      mac     y1,y0,b   y:(r4)+,y0
      macr    x0,y0,b   y:(r4)+,y0
      mpy     y1,y0,b   b,x:(r0)+   y:(r4)+,y0
      macr    x0,y0,b   x:(r0),a    a,y1
      asr     a         b,x:(r1)+   y:(r4)+,y0
_end
```

Since the digital filter is realized on a fixed-point digital signal processor, we set the state space variables to the limit values of the complementary code range. We have assumed that the structure chosen is robust enough and that the response to initial conditions will abate quickly and that no problems will appear. However, in spite of the suitably chosen structure, correctly set coefficients of the partial sections, etc., we got to the limit of implementation stability, that is to say the effects of quantizing the coefficients, of partial results of mathematical operations within the arithmetic-logic unit of digital signal processor, etc., result in that output signal y(n) does not get steady and remains chaotic. See [5] for more details.

2.2. Optimizing Modulation Effect Algorithms

The fundamental element of all modulation effects is a delay line with variable delay and a low-frequency oscillator (LFO) that controls the delay of this delay line.

2.2.1. Delay Line with Variable Delay

The delay line with variable delay can be realized 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 latency time τ is given by the relation

$$\tau(n) = Depth \cdot g(n), \tag{6}$$

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:

$$K = \left|\tau(n) \cdot f_S\right|. \qquad (7)$$

However, multiplication by the value of sampling frequency cannot be realized in the fixed-point signal processor. Therefore we have to split the sampling frequency value into the product of a number <1 and a power of 2. Using the bit-rotation instructions we place the integer part of number *K* into the *a1* accumulator part and the fractional part into the *a0* accumulator part. The integer part of number *K* gives information about the position of two neighbouring samples in the buffer between which interpolation will be performed. Buffer length *L* depends on the required maximum delay time τ_{MAX} and sampling frequency f_S:

$$L = \tau_{MAX} \cdot f_S \qquad (8)$$

With the regard to the possibilities of the Motorola DSP address generation unit the buffer length of a power of 2 is useful. If we choose a buffer length of 213, i.e. 8192 samples, and a sampling frequency of 48 kHz, the maximum line delay will be about 170 ms, which is sufficient for all common applications.

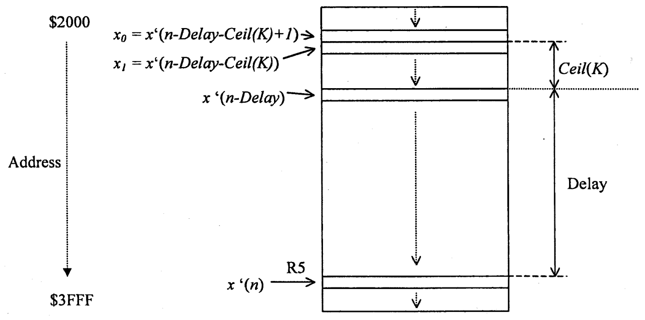

$2000
$x_0 = x'(n\text{-}Delay\text{-}Ceil(K)+1)$
$x_1 = x'(n\text{-}Delay\text{-}Ceil(K))$
$x'(n\text{-}Delay)$
Ceil(K)
Address
Delay
R5
$x'(n)$
$3FFF

Figure 3. *Modulo buffer*

We have to carry out the interpolation of two neighbouring samples to obtain output sample x_D:

$$x_D = x_0 \cdot \left(1 - fract(K)\right) + x_1 \cdot fract(K), \qquad (9)$$

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

$$x_0 = x'\left(n - Delay \pm ceil(K)\right) \qquad (10a)$$

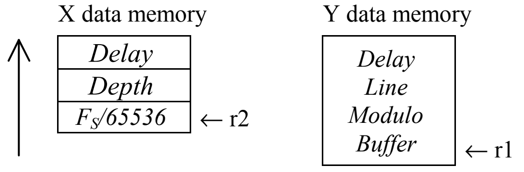
$$x_1 = x'\left(n - Delay \pm ceil(K) \pm 1\right), \qquad (10b)$$

where *Delay* is the adjusted mean magnitude of delay and *ceil(K)* is the integer part of *K*. Important relations among modulation depth, mean value of delay, and modulo buffer size *L* follow from Fig. 3:

$$Delay + Depth < L \qquad (11a)$$
$$Delay > Depth. \qquad (11b)$$

Algorithm needs 2 addreses in the X data memory, where delay line parameters – mean value of delay and modulation depth - will be stored, and 1 address position in the Y data memory for storing information about the sampling frequency.

Memory usage:



Evaluation of delay amount (*y1* contains relative delay):

```
      move             l:(r2)+,x
      mpyr   x0,y0,a   x:(r2),x0
      move             a,y0
      mpy    x1,y0,a   x0,n1
      abs    a         a,b
      rep    #7
      asr    a         r5,r1
      move   a1,n5
      move   a0,a1
      asr    a
      move   a1,y0
      move   n5,n1
```

Obtaining two samples for interpolation (*x0* and *x1* registers):

```
      tst    b
      move             (r1)+n1
      move             y:(r1)+,x0
      move             y:(r1)-,x1
      jmi    _nxt
      move             (r1)-n1
      move             (r1)-n1
      move             y:(r1)-,x0
      move             y:(r1),x1
```

Interpolation (at the end of algorithm, *b* contains the output sample):

```
_nxt  mpy    x1,y0,b   #$3fffff,a
      asl    a
      sub    y0,a
      move   a,y0
      mac    x0,y0,b
```

2.2.2. *Low Frequency Oscillator*

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

$$g(n) = g(n-1) + k \quad \text{for the leading edge} \qquad (12a)$$
$$g(n) = g(n-1) + (-k) \quad \text{for the trailing edge,} \qquad (12b)$$

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

$$k = \frac{4}{f_s} \cdot f_{LFO}, \qquad (13)$$

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 (12a). Since the duration of the leading edge of the saw waveform is twice that of the triangular waveform, the increment for the saw waveform must be half the increment in (13). 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. 4.

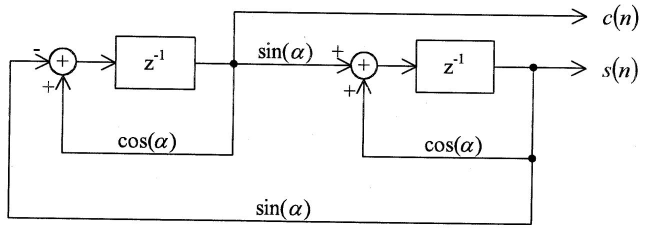


Figure 4. *Recursive generator of sine and cosine waveforms*

The realization equations of the recursive generator from Fig. 4 are

$$c(n+1) = c(n).\cos(\alpha) - s(n).\sin(\alpha) \tag{14a}$$

$$s(n+1) = c(n).\sin(\alpha) + s(n).\cos(\alpha)\,, \tag{14b}$$

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

$$\alpha = 2\cdot\pi\cdot\frac{f_{LFO}}{f_S}\,, \tag{15}$$

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 values of generated waveforms are placed into the X data memory. Algorithm parameters (initialization values and increment) and memory pointer of currently selected waveform are stored in the Y data memory. Saving the new pointer that points to the newly selected waveform does the LFO waveform selection.

Memory usage:

X data memory		Y data memory	
rectangle		*Selected wave adr.*	
triangle		*$800000*	
saw		*4*F1/Fvz*	
sine		*@sin(2*pi*F1/Fvz)*	
cosine	← r0	*@cos(2*pi*F1/Fvz)*	← r4

Sine and cosine waveform:

```
move              x:(r0)+,x0   y:(r4)+,y0
mpy    x0,y0,a    x:(r0)+,x1   y:(r4)+,y1
macr   -x1,y1,a
mpy    x0,y1,b    a,x0
macr   x1,y0,b    x:(r0)-,a
```

Saw waveform:

```
move   a,b     b,x:(r0)-    y:(r4),b
addr   a,b     x0,x:(r0)+   b,y1
clr    a       b,x:(r4)+
tfr    y1,b    x:(r4)+a     a,y0
```

Triangle and rectangle waveform:

```
     jclr   #23,x:(r4)-,_sub
     neg    b
_sub add    b,a
     jec    _of
     neg    b         y0,x:(r0)-
     add    b,a       y:(r4),b
     jpl    _inc
     move             b,x:-(r4)
     neg    b         (r4)+
_inc move             b,x:(r0)+    y:(r4)+,y0
     move             b,x:(r4)-
_of  move             a,x:(r4)+    y:(r0)+,y0
     bclr   #6,sr
```

2.3. Optimizing Non-linear Effect Algorithms

The basic block of non-linear effect algorithms is a transfer block with non-linear transfer characteristic. In our algorithms we use the non-linear transfer characteristic according to Fig. 5 (see [6] for details).

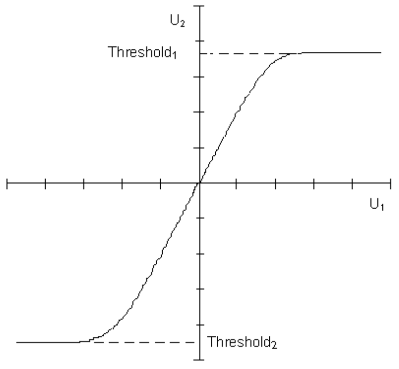


Figure 5. *Static transfer characteristic of non-linear system*

Signals outside the level range defined by the *Threshold1* and *Threshold2* values are limited. Part of the transfer characteristic between the *Threshold1* and *Threshold2* values is approximated by a power polynomial.

Features of the saturation arithmetic of Motorola DSP56k digital signal processor, which limits numbers greater than 1-2-24 and less than –1 during the transfer from accumulator to data bus, can be used for signal limitation. First we make the transfer characteristic symmetric by adding the c_{offset} constant

$$c_{offset} = -\left(Treshold_1 + Threshold_2\right) \tag{16}$$

and then we transfer this characteristic into the <-1,1) range by multiplying by the c_{range} constant:

$$c_{range} = \frac{2-2^{-24}}{Treshold_1 - Threshold_2}\,. \tag{17}$$

The modified input signal *x'(n)* will be

$$x'(n) = c_{range} \cdot x(n) - c_{offset} \, . \tag{18}$$

We divide the multiplication by the c_{range} constant into the multiplication by a number that is a power of 2 and a number less than 1. Equation (18) then changes to the form:

$$x'(n) = 2^{c_{rangeP}} \cdot c_{rangeF} \cdot x(n) - c_{offset} \, . \tag{19}$$

The calculation of the polynomial that approximates the characteristic between the *Threshold1* and *Threshold2* points follows the input signal modification. The common polynomial equation is

$$y = a_K x^K + a_{K-1} x^{K-1} + \ldots + a_2 x^2 + a_1 x + a_0 \, . \tag{20}$$

Equation (20) can be modified to the form

$$y = a_0 + a_K \prod_{i=1}^{K} x + a_{K-1} \prod_{i=1}^{K-1} x + \ldots + a_1 \prod_{i=1}^{1} x \, . \tag{21}$$

If we introduce state variables

$$v_0 = 1 \, , \; v_k = x.v_{k-1} \tag{22}$$

we can write equation (21) in the form

$$y = a_0 + a_1 v_1 + \ldots + a_K v_K = a_0 + \sum_{i=1}^{K} a_i v_i \, . \tag{23}$$

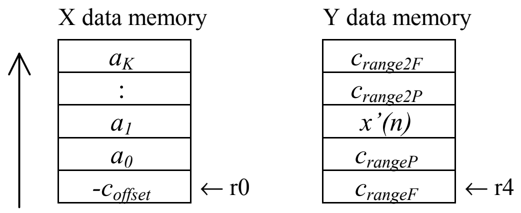
We obtain the output signal using the equation

$$y(n) = y'(n) \cdot 2^{c_{range2P}} c_{range2F} + c_{offset} \, , \tag{24}$$

where $c_{range2P} = -c_{rangeP}+1$, $c_{range2F} = 1/(2.c_{rangeF})$ and *y'(n)* is the modified output signal obtained by calculating the polynomial from modified input signal *x'(n)*.

We can realize equation (23) on the Motorola DSP56k digital signal processor using a modulo buffer of length K and the instructions multiply with accumulate and multiply closed in the DO cycle. The algorithm computing power is 3K+1 instructions (without initialization) and the algorithm requires K address positions in data memory.

Memory usage:



Input signal modification (*y1* contains input sample):

```
move              x:(r0)+,b   y:(r4)+,y0
macr   y0,y1,b    x:(r0)+,a   y:(r4)+,x0
rep    x0
asl    b          #2,n4
tfr    b,x1       x:(r0)+,x0  b,y:(r4)
```

Polynomial evaluation:

```
      do     #K,_end
      mac    x0,x1,a   y:(r4),x0
      mpy    x0,x1,b   x:(r0)+,x0
      move   b,x1
_end
```

Output signal evaluation (at the end, *b* contains the output sample):

```
      move             x:(r0),b   y:(r4+),x0
      rep    x0
      asr    a         a,y0       y:(r4+n4),y1
      mac    y0,y1,b   (r4)-n4
```

3. CONCLUSION

This paper presents the usage possibilities of DSP architecture features to optimize digital musical effect algorithms. The utilization of the Motorola DSP56k family architecture is shown on three concrete examples of basic real-time digital musical effects, which work in the time domain – double Harvard architecture (up to two parallel moves within an instruction execution), Arithmetical-Logic Unit (integrated hardware multiplier and saturation arithmetic), and Address Generation Unit (modulo buffers and simultaneous work with two data memories). Such optimized algorithms save the DSP power, which is then able to process more plug-in effects at the same time. The DSPs can take on more processing of digital musical effects in the DAW system and thus save the host PC computing power for other processes, which run in parallel, for example hard disc recording, MIDI data processing, etc.

REFERENCES

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

[2] Virtual Studio Technology Plug-In Specification 2.0 Software Development Kit. Steinberg Soft- und Hardware GmbH. 2000.

[3] DSP56000 Digital Signal Processor Family Manual. Motorola Inc., USA, 1993.

[4] Oboril, D., Balik, M., Schimmel, J., Smekal, Z., Krkavec, P. Modelling Digital Musical Effects for Signal Processors, Based on Real Effect Manifestation Analysis. Proceedings of COST-G6 conference on Digital Audio Effects DAFx00, Verona, December 7 to 9, 2000, pp. 165 – 170. ISBN 88-900547-0-0

[5] Smekal, Z. "Optimum Digital Filter Structure Design Based On Response To Initial Conditions" [online], Brno 2000, [20.12.2000]. ISSN 1213-161X. Available at URL <www.electronicsletters.com/papers/smekal/smekal.asp>

[6] Smekal, Z. et al.: Partial research report on the solution of international project No OC G6 10 for the year 2000.