Audio Engineering Society

Convention Paper 6202

Presented at the 117th Convention
2004 October 28–31 San Francisco, CA, USA

Frequency-Domain Additive Synthesis With An Oversampled Weighted Overlap-Add Filterbank For A Portable Low-Power MIDI Synthesizer

King Tam[1]

[1] Dspfactory Ltd., Waterloo, Ontario, N2V 1K8, Canada
king.tam@dspfactory.com

ABSTRACT

This paper discusses a hybrid audio synthesis method employing both additive synthesis and DPCM audio playback, and the implementation of a miniature synthesizer system that accepts MIDI as an input format. Additive synthesis is performed in the frequency domain using a weighted overlap-add filterbank, providing efficiency gains compared to previously known methods. The synthesizer system is implemented on an ultra-miniature, low-power, reconfigurable application specific digital signal processing platform. This low-resource MIDI synthesizer is suitable for portable, low-power devices such as mobile telephones and other portable communication devices. Several issues related to the additive synthesis method, DPCM codec design, and system tradeoffs are discussed.

1. INTRODUCTION

Additive synthesis in musical applications has been known in the field for many years. The method is based on the concept that complex musical sounds can be generated by summing multiple components that are relatively simple on their own [1, 2]. Commonly, the summed components are sinusoids, referred to as *partials*. The model of sound as a summation of sinusoids with time-varying frequency and amplitude is intuitive, and also maps well to analysis and implementation using the Fourier transform and inverse Fourier transform.

While several other synthesis methods have been developed, interest in additive synthesis has continued. Additive synthesis is especially interesting in applications where the processing power and memory required by other methods are prohibitive because of limited system resources (power, memory, computational speed). These applications are abundant, particularly given the proliferation of mobile battery-powered devices such as mobile phones, gaming devices, and personal electronic organizers.

The method described in this paper uses a hybrid of additive synthesis and playback of audio samples. The design of the synthesizer system gives special consideration to computational efficiency and limited availability of processor memory.

Figure 1 shows an overview of the processing performed by the synthesizer system.

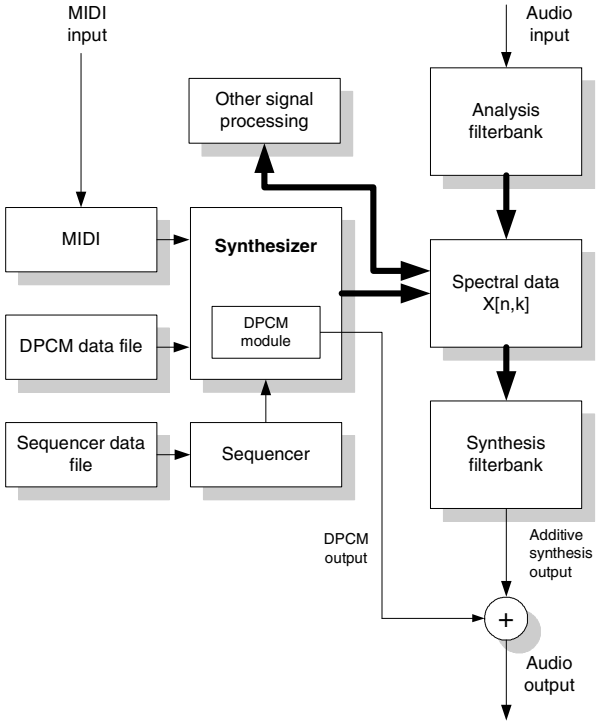

Figure 1: Synthesizer System Block Diagram

The remaining sections of this paper are organized as follows. In section 2, the additive synthesis method is described; then, in section 3, the DPCM codec is discussed. Section 4 describes the operation modes of the synthesizer. Section 5 presents the hardware platform used. Finally, sections 6 and 7 discuss the additional processing that may be added, and the tradeoffs that are made.

2. FILTERBANK SYNTHESIS

2.1. WOLA Filterbank

A frequency domain approach for additive synthesis is taken, utilizing a weighted overlap-add (WOLA) filterbank [3] based on the forward and inverse discrete Fourier transform. A musical note is formed by synthesizing a number of summed partials that are described in the transform domain. A significant gain in efficiency is obtained from the fact that the spectral data is represented at a decimated sampling rate, instead of at the full rate of the synthesizer output. Of course, there is some overhead cost associated with the filterbank processing; however, it is dependent only on the filterbank parameters (transform size, window length, etc.) and not on the number of partials.

The WOLA filterbank has the property that the transformed signal is decomposed into subband signals (i.e., the spectral data) with a low degree of frequency overlap between adjacent subbands. As a result, a pure-tone input to the analysis filterbank results in subband decomposition with a significant component in only one subband, and very low-level components in the adjacent subbands. Therefore, when generating data for synthesis, one partial is efficiently represented by only one non-negligible spectral component. In contrast, other methods represent a partial using more than one spectral component, and therefore have additional computational and memory storage costs associated this more complex representation. For example, the method described by Rodet and Depalle typically uses 9 spectral components per partial [4].

2.2. Calculating Partials

The time-varying spectral component for a partial is calculated by first determining the index of the subband into which the non-zero component will fall. The WOLA analysis and synthesis filterbank has $N/2$ uniformly spaced subbands, each having a width of f_s/N, covering the frequencies between 0 and $f_s/2$, where N is the Fourier transform size and f_s is the sampling rate. The subband index for the *qth* partial, k_q, is determined by finding the subband that has a center frequency closest to the frequency of the partial. If the partial frequency is exactly at the midpoint between the center frequencies of two adjacent subbands, either of the two can be arbitrarily chosen.

Let $P_q[n,k]$ be the complex-valued spectral component associated with the *qth* partial at time index n (i.e., the decimated subband time index) for subband index k,

and $V_w[n,k]$ be the spectral data for the complex sound instance (referred to here as a *voice*) *w* at time *n*. Then

$$P_q[n,k] = \begin{cases} s_q[n] \cdot e^{j \cdot \theta_q[n]} & , \; k = k_q \\ 0 & , \; k \neq k_q \end{cases} \quad (1)$$

and

$$V_w[n,k] = a_w \cdot \sum_q P_q[n,k] , \quad (2)$$

where a_w is the amplitude scalar associated with the volume of the voice.

The spectral data $X[n,k]$ finally passed to the synthesis filterbank is the sum of $V_w[n,k]$ for all the active voices:

$$X[n,k] = \sum_w V_w[n,k] \quad (3)$$

In equation (1), $s_q[n]$ is the aggregate envelope function (discussed in section 2.4) associated with the partial, and $\theta_q[n]$ is the phase angle which can be calculated using the following equation:

$$\theta_q[n] = \theta_{q,0} + \omega_q \cdot n , \quad (4)$$

where $\theta_{q,0}$ is the initial phase. As a simplification, we let $\theta_q[n] = 0$ when the partial initially becomes active (a synthesizer voice is started), and then use the following relationship for the rest of the duration:

$$\theta_q[n+1] = \theta_q[n] + \omega_q \quad (5)$$

The angular frequency ω_q is determined by the following equation:

$$\omega_q = \left[\frac{f_q - f_c(k_q)}{f_s / N} \right] \cdot \left[\frac{2\pi}{OS} \right] \; rad / sample , \quad (6)$$

where f_q is the frequency of the partial, $f_c(k_q)$ is the center frequency of subband index k_q, and OS is the filterbank oversampling ratio.

Equation (6) is intuitively satisfying since it can be seen that ω_q will equal zero if the frequency of the partial falls exactly on the center frequency of the subband (the synthesis filterbank will modulate the 0 Hz spectral component to $f_c(k_q)$). At the other extreme, if the frequency of the partial lays exactly between the center frequencies of two adjacent subbands (i.e., $f_q - f_c(k_q) = 0.5 \cdot f_s / N$), then ω_q equals π / OS.

The WOLA filterbank oversampling ratio OS is the factor to which the decimated sampling rate of the subband signals is greater than the critically decimated rate (i.e., the theoretical minimum rate given the width of the subbands). It is determined by the ratio of the Fourier transform size N to the input/output block size R. Typically N/R equals 4 or 2, and therefore the sampling rate of the subband signals (i.e., the rate of update for the spectral components of the partials) is 4 or 2 times greater than the critical rate. The factor OS is included in the expression for ω_q to compensate for this oversampling.

2.3. Defining Sounds

The description of an "instrument" sound (or *patch*) is stored by the synthesizer system as a set of partials along with an envelope specification field. Then each partial has three characteristics associated with it: frequency, amplitude, and envelope. The frequency is expressed in relative terms with respect to the fundamental frequency of the musical note. The amplitude specifies the energy of the partial relative to the others used for the instrument sound. Finally, the envelope is specified for the partial, as described in section 2.4.

Naturally, the definition of the instrument sounds in terms of the relative frequencies and amplitudes of the partials plays an important role in the perceptual quality of the synthesized sound. The timbres of the synthesized sounds may or may not be typical of real instruments. For emulation of real instruments, offline analysis of recorded sounds can be used to derive the instrument definitions for the synthesizer.

2.4. Envelope Functions

An envelope function can be defined in memory as an arbitrary waveform in a table, allowing for a large

degree of flexibility at the cost of a relatively large amount of memory usage. Another class of envelopes is described only as parameters for linear or exponential functions. This requires far less memory and increases the computational load slightly since the envelope functions must be calculated during execution.

Note that envelopes are designated at both the instrument level and for each individual partial, and storage of the envelopes (for the type stored as waveforms in memory) is not repeated although there may exist repeated references to the same envelope. Referring back to equation (1), the aggregate envelope $s_q[n]$ function is formed from the product of the envelopes corresponding to the instrument sound and the specific partial, and the partial's relative amplitude.

2.5. Lookup Tables

To calculate $e^{j\cdot\theta_q[n]} = \cos\theta_q[n] + j\cdot\sin\theta_q[n]$ in equation (1), we first calculate $\theta_q[n]$ using equation (5). Then $\theta_q[n]$ is used to index a cosine and sine lookup table. A length of 360 elements for the sine table gives adequate granularity without occupying too large a portion of memory. Appending 90 words to the sine table and appropriately offsetting the lookup index implements the cosine table that can be addressed linearly.

3. DPCM DECODER

After the synthesis filterbank uses $X[n,k]$ to generate the time-domain output frame of the additive synthesizer, this output is added to the output of the DPCM decoder. DPCM playback is employed in addition to the filterbank-based synthesis method in order to enhance the range of sounds available because the additive synthesis is not well suited for some sounds (e.g., noise, transient sounds). The synthesizer stores compressed snare drum, bass drum, and cymbal samples for on-demand decompression and playback.

The DPCM format is chosen because of the relative simplicity of the decoder and acceptable data compression ratio of 4:1. It is especially important in mobile applications to compress data where possible in order to minimize the system memory requirements [5].

Encoded audio samples are stored in memory, then decoded and played back when appropriate. Figure 2 shows the encoder and decoder used.

A first-order linear predictor is used in order to limit the decoder complexity and execution time. The predictor output $\hat{x}[t]$ is simply the input with a one-sample delay: $\tilde{x}[t-1]$.

The quantizer block of the DPCM encoder is optimized offline using the Generalized Lloyd Algorithm (GLA) in an effort to minimize the slope overload distortion and noise introduced by the coding of the audio samples [6]. Using the drum samples as input data, the GLA determines the set of quantization levels which result in low distortion, measured by the difference between input and output of the quantizer. Once this process has determined the quantization levels to be used for the selected encoder inputs, the levels are stored in the DPCM decoder and used during runtime. The GLA is required again only if the stored audio samples change and the DPCM codec requires offline optimization of the quantizer once more.

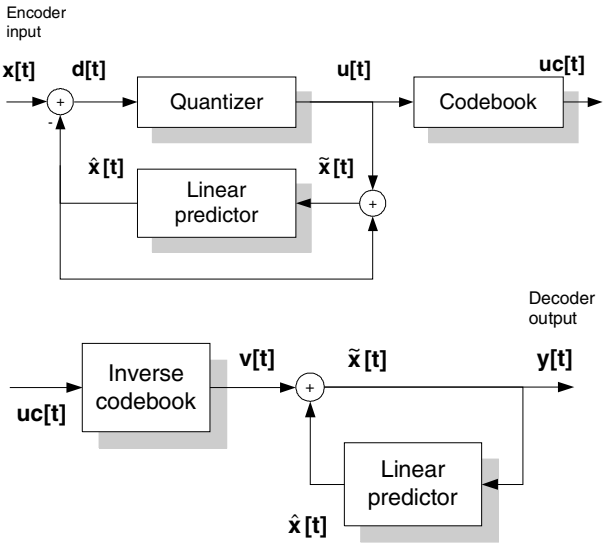

Figure 2: DPCM Encoder and Decoder

4. OPERATION MODES

The synthesizer system has two modes of operation:

- playback mode: playback from a file stored in memory by the sequencer
- MIDI mode: real-time input through MIDI

In playback mode, the sequencer block is activated and it controls the synthesizer. When real-time input is being processed, the MIDI block becomes active and processes an external device's incoming commands.

The mode of operation is set according to the target application using firmware settings.

4.1. Playback Mode

The sequencer block of the synthesizer system is responsible for controlling the time evolution of the output in playback mode. It interprets a stored data file that describes a collection of musical notes to be played (for example, ringtones used for a mobile phone). Commands are then sent to the main synthesizer block and DPCM decoder to start and stop the voices at the appropriate times, with the correct instrument sounds or DPCM samples, and with the desired volume setting.

The sequencer data file is generated offline, either through manual entry or through the use of automated tools. These tools have been developed to aid in the translation of standard MIDI files (SMF) to the sequencer's internal file format. Some effort has been made to make this format translation simple by choosing a command set that has some similarities with SMF.

The sequencer supports note-on, note-off commands, as well as some commands to set the tempo. Simple looping is also supported.

4.2. MIDI Mode

In MIDI mode, the input stream from an external device must be processed in real-time by the synthesizer system. The MIDI input is processed by the same processor that performs the synthesis.

5. HARDWARE PLATFORM

The synthesizer system is implemented on an ultra low-power miniature digital signal processing (DSP) platform. It consists of DSP capability in addition to analog-to-digital converters, digital-to-analog converters, and some peripheral interfaces, as shown in Figure 3.

The digital synthesis operations are performed by a programmable DSP core and a reconfigurable coprocessor that executes in parallel with the core. WOLA filterbank operations are the main task of the coprocessor, while the more flexible DSP core handles the rest of the processing including handling of the serial data interface for MIDI.

Another unit called the IOP (Input/Output Processor) automatically handles the movement of audio samples between the input and output FIFOs and the analog-to-digital and digital-to-analog converters.

This parallel processing hardware architecture facilitates the efficient implementation of the methods described in this paper.

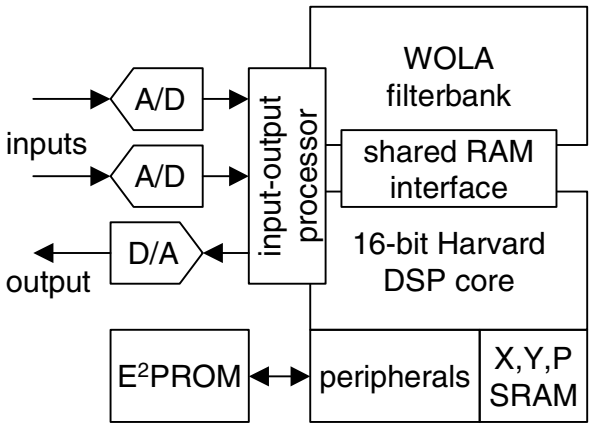

Figure 3: DSP Hardware Platform

The system contains 8 kwords of data memory and 12 kwords of program memory, and has an 8 x 8 mm QFN package size (without EEPROM).

As with all MIDI ports, some circuitry is necessary to perform opto-isolation. This is achieved through some simple external circuitry that is connected to the UART port of the DSP hardware platform. The size of this circuit and the connector for MIDI is not included in the package size stated above.

For this application the system is clocked at 24 MHz, and the sampling rate is set to 22 kHz. The power consumption is approximately 6 mW when the system is

active. This does not include power drawn for the loudspeaker or microphone that may be connected to the input stage.

6. ADDITIONAL PROCESSING

The system accepts both analog and digital signal inputs for processing and mixing with the synthesizer output. A number of different types of subband processing algorithms benefit from the properties of the WOLA filterbank and parallel system architecture [3, 7, 8, 9]. The programmable nature of the platform allows for the integration of other signal processing blocks that operate on the input signal or synthesized signal, or both.

An important property of the WOLA filterbank is the orthogonality of the subband signals, which makes independent processing in each subband possible without objectionable distortion present in the filterbank output. For example, for multi-channel dynamic range compression, this allows a higher degree of possible gain variation between adjacent subbands [9].

A wide variety of other processing algorithms – including noise reduction, equalization, and acoustic feedback control – are also well suited for implementation on this system and integration with the synthesizer; however, the details of these processing blocks are outside the scope of this paper.

7. TRADEOFFS

Given the limited resources available in terms of computational speed and memory, a number of important tradeoffs are made that determine the synthesizer's capabilities.

The number of simultaneous voices that the synthesizer is capable of is 8, the maximum number of partials per voice is 16, the number of instrument sounds is 4, and 3 DPCM samples are stored. Three envelopes of the waveform table type are supported (parametric envelopes are preferred for their efficient memory usage). The size of the sequencer file is limited by the amount of memory taken by the other components of the system. These settings result in a good sound quality that is well suited for portable applications.

The programmable nature of the DSP platform gives the flexibility that allows the tradeoffs to be adjusted depending on the target application. As an example, more memory may be allocated for DPCM samples and envelopes at the cost of less memory for the sequencer file.

Beyond adjustments to the firmware, the system may be clocked at a higher rate to increase the processing cycles available, perhaps in order to increase the degree of polyphony. Alternatively, the clock rate may be decreased to lower the power consumption at the expense of less processing capability.

8. SUMMARY

This paper presents a new memory-efficient and power-efficient hybrid musical synthesis method employing additive synthesis using a WOLA filterbank and DPCM playback. The implementation of these methods on the DSP hardware platform results in a miniature synthesizer system suitable for portable applications.

The synthesizer system runs on a 24 MHz system clock and is configured for a sampling rate of 22 kHz. It contains 8 kwords of data memory and 12 kwords of program memory, has a package size of 8 x 8 mm without EEPROM, and consumes approximately 6 mW (not including transducers) when active.

Two operation modes are supported: playback of a song or tune (for example, a ringtone) stored in a file in memory, and real-time processing of MIDI input.

The implementation allows the flexibility to make adjustments to the capabilities of the synthesizer, and to integrate other processing algorithms that also use the WOLA filterbank.

9. ACKNOWLEDGEMENTS

The author would like to thank Etienne Cornu and Todd Schneider for their valuable input and helpful comments.

10. REFERENCES

[1] J. C. Risset, M. V. Mathews, "Analysis of Musical-Instrument Tones" Physics Today, vol. 22, no. 2. Feb. 1969.

[2] F. R. Moore, *Elements of Computer Music*, Prentice Hall: Upper Saddle River, NJ. 1990.

[3] R. Brennan, T. Schneider, "A Flexible Filterbank Structure for Extensive Signal Manipulations in Digital Hearing Aids", Proc. IEEE Int. Symp. Circuits and Systems, pp.569-572, 1998.

[4] X. Rodet, P. Depalle, "Spectral Envelopes and Inverse FFT Synthesis", 93rd AES Convention, San Francisco, 1992.

[5] R. C. Maher, "Compression and Decompression of Wavetable Synthesis Data", 115th AES Convention, New York, 2003.

[6] N. S. Jayant, P. Noll, *Digital Coding of Waveforms*, Prentice Hall: Englewood Cliffs, NJ. 1984.

[7] K. Tam, H. Sheikhzadeh, T. Schneider, "Highly oversampled subband adaptive filters for noise cancellation on a low-resource DSP system", Proc. 7th Int. Conf. on Spoken Language Processing (ICSLP), Denver, CO, September 16-20, 2002.

[8] J. Johnson, E. Cornu, G. Choy, J. Wdowiak, "Ultra Low-Power Sub-Band Acoustic Echo Cancellation For Wireless Headsets", Proc. ICASSP 2004.

[9] T. Schneider, R. Brennan, "A Multichannel Compression Strategy For A Digital Hearing Aid", Proc. ICASSP 1997.