Implementation and Hardware Approach to MIDI-Synthesizer using DSP TMS320C6713

Emad M. Arasté
Department of Electrical and Information Technology
Lund University
Lund, Sweden
Email: sx08em2@student.lth.se

Iman Mohajeri
Department of Electrical and Information Technology
Lund University
Lund, Sweden
Email: px08im6@student.lth.se

***Abstract*—Music synthesis is a method for generating sound for different musical instruments through various digital or analog techniques. The aim of this work is to provide a DSP solution as well as hardware approach to MIDI-Synthesizer. We employed Frequency Modulation and Wavetable methods which both are well known techniques for creating different notes of musical instruments like piano, strings and brass. We made usage of MATLAB to investigate frequency spectrum of notes for different instruments and then we tried to apply FM technique and Wavetable method to implement 8 instruments which 6 of them have 4-polyphony property. The code is written in C and developed by Code Composer studio and is implemented in real time using the Texas Instruments DSP board DSK6713.**

I. Introduction

From the production of Yamaha DX7 as the first digital synthesizer and introducing MIDI (Musical Instrument Digital Interface) protocol in mid of 80s, still huge efforts are being made to produce low-power and highly efficient MIDI-Synthesizers.

FM (Frequency Modulation) method and Wavetable synthesis are commonly used techniques in synthesizers at present. The FM synthesis method has a privilege in generating various range of sounds but lack of sound performance; and wavetable synthesis method has a better quality in sound, however it makes some limitations in memory usage of chip [1].

MIDI file consists of commands that say when a note should be played on a certain channel and when it should be stopped. A synthesizer decodes these commands that mostly are MIDI events. Each MIDI event has 5 distinct parts: delta time, event type, MIDI channel, note number and velocity [2].

TMS320C6713 DSK board can be used as a proper platform to implement MIDI-Synthesizer. Advanced very-long-instruction-word (VLIW) architecture of C6713 provides a good basis for multichannel applications.

II. Digital synthesis techniques

Implemented synthesizer enjoys less complexity in computation and acceptable quality in sound thanks to combination of FM and wavetable techniques. In the following subsections, we will describe 2 mentioned techniques and in the last we will express the essence of envelope for music synthesis.

A. Frequency Modulation

Synthesis by frequency modulation was initiated by John Chowning in the late 60s. The idea for FM synthesis is based on the same idea used for FM radio transmission. The FM-modulated signal with carrier frequency of F_c and modulation frequency of F_m is [3] :

$$S(t) = A(t)cos(2\pi F_c t + I(t)cos(2\pi F_m t + \phi_m) + \phi_c) \quad (1)$$

Where ϕ_m and ϕ_c are initial phases and are assumed $\pi/2$ in this papar. Carrier frequency defines main frequency of signal and modulation frequency determines sideband frequencies. Sideband picks are always placed symmetrically around the main frequency in F_c+nF_m and F_c-nF_m where *n* is an integer. In order to determine the sidebands which are present in frequency spectrum of signal, we have to control ratio between the carrier frequency *(F_c)* and the modulation frequency *(F_m)*. Therefore, we will use *C:M* ratio instead of expressing these frequencies in Hz. *I* is a modulation index, which specifies how much energy should be distributed to sidebands and defines amplitudes of sideband frequencies

B. Wavetable

Wavetable synthesis employs stored samples in memory to produce the sound. It is achieved by importing the quantized values of real sound to memory and forming a lookup table [4] [5]. One simple algorithm to exploit an L-sample table without rescanning the table could be as follow:

$$\text{sample\_value} = \text{lookup\_table[index]} \quad (2)$$

$$\text{output} = \text{AMP} \times \text{amplitude} \quad (3)$$

where *sample_value* is the magnitude of the sound wave sample at instant *index* and *AMP* corresponds to normalized amplitude of signal and *index* is always less than L. Rapid increase in memory capacity along with less cost in manufacturing granted broad usage of wavetable technique in roughly every current produced synthesizer.

C. Envelope

Since the notes do not start and stop instantaneously, knowing just the information in MIDI file will not be sufficient for generating a realistic sound. It takes a finite time for a string to start vibrating, and time for it to degrade to a stable state [6]. Therefore, we need a pattern to change the volume of produced sound during note play time. This pattern is called envelope. Each instrument has its own volume characteristic; consequently, it needs special envelope. The most common used envelope is ADSR-envelope, which is abbreviation of Attack (A), Decay (D), Sustain (S) and Release (R). (Fig. 1)

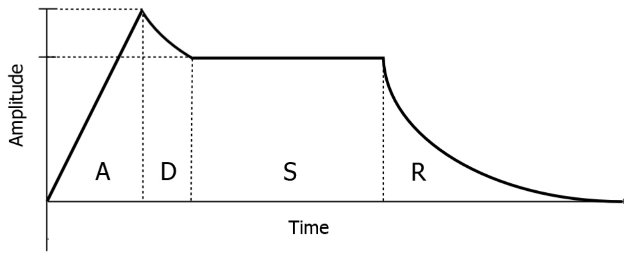

Fig. 1. ADSR-envelope

Attack time is the time at which sound reaches to its maximum volume after it is activated. Usually, this time is very short for piano. Decay time is from when the sound starts dropping from its full volume to an almost constant value. Sustain is a period of time which the volume of sound is nearly constant. It starts after decay and ends before note is finished. Release shows how fast sound should fade away. Usually, this time is very short except in some instruments like piano in case the pedal is pressed.

III. Simulation

Having a good quality sound calls exact inspection of generated waveform of each musical instrument. This section comprises the investigation of frequency spectrum of three musical instruments including bell, brass and piano with the aid of MATLAB and acquisition of drum sound using GarageBand.

First, recorded single note sound of each mentioned instrument imported to MATLAB were sampled with sampling frequency of 8KHz and 16-bit sampling precision. Drawing the time domain waveform of a base note for each instrument made the inspection of envelope practical.

Bell sound is realized by assuming musical note frequency as the carrier and *C:M* ratio equal to 0.5. We defined the modulation index (*Ienv*) and amplitude (*Aenv*) based on equation (4) and (5).

$$\text{Aenv} = A_0 \times e^{-t/\tau} \tag{4}$$

$$\text{Ienv} = A_0 \times e^{-t/\tau} \tag{5}$$

where τ is time constant.

Brass sound is made by frequency ratio of 1 and modulation index of 5. As illustrated in Fig. 2 brass envelope has a linear property which makes it quite easy to implement.

Among different instruments, piano is the hardest to implement, since there is a complicated physical process behind it.

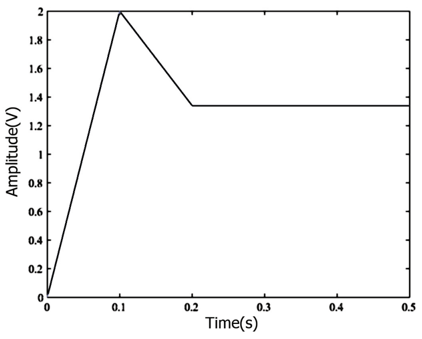

Fig. 2. Brass Envelope

By extracting the frequency spectrum of the recorded sound of a base note and assigning different values for FM parameters, we deduced the following equation to produce the piano sound:

$$\text{Aenv} = A_0 \times t \tag{6}$$

$$\text{Aenv} = A_0[e^{(t_i - t)} + A_i] \tag{7}$$

where equation (6) and equation (7) correspond to attack and sustain parts respectively. Also t_i and A_i are constant values. Fig. 3 depicts the comparison between frequency spectrum of both the real and the simulated piano C5 note.

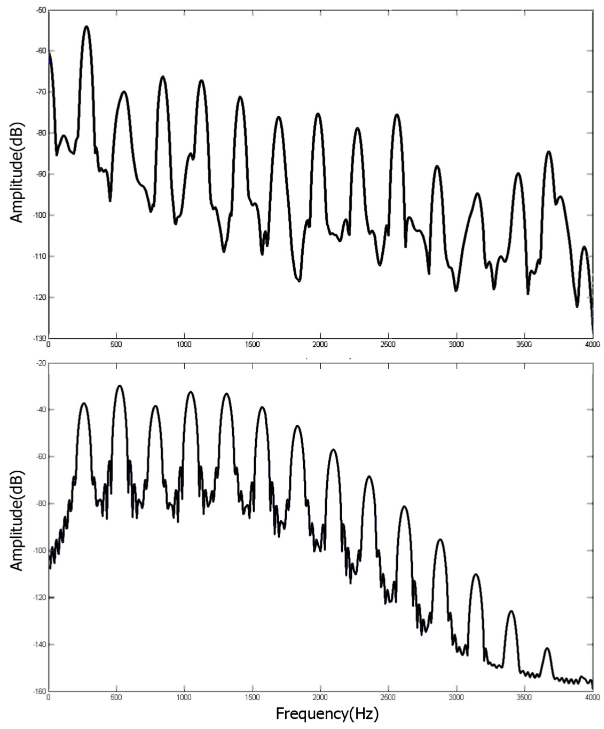

Fig. 3. Frequency spectrum of real (*top*) and simulated (*bottom*) C5 piano note

On the other hand, we needed an envelope to apply on the wave form. From the waveform we discovered that the piano sound amplitude degrades as an exponential function, so we tried to calculate the corresponding exponential function in accordance to the real waveform. The simulated envelope and piano C5 note waveform is shown in Fig. 4. The attack and sustain parts are clear in the figure. For the attack section a linear equation is used whereas the sustain part is achieved by an exponential equation:

$$\text{Aenv} = A_0 \times (e^{-t+ti} + A_i) \qquad (8)$$

$$\text{Ienv} = 3.5 \qquad (9)$$

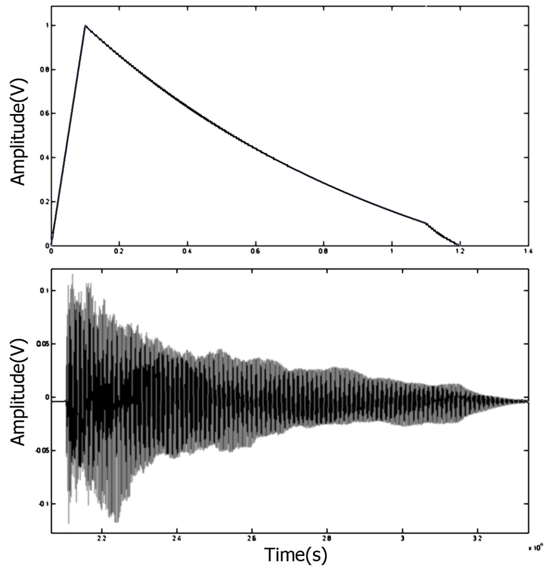

Fig. 4. Implemented piano envelope (*top*) and waveform of C5 piano note (*bottom*)

The last instrument which is partially effective in music quality is drum. A good drum makes music so rich and effective; therefore we tried to implement the drum with wavetable synthesis method. Seven drum notes generated by GarageBand and sampled with MATLAB, are included in the form of header files in the implementation part. These samples could also be loaded to DSP in setup phase from external memory.

IV. Verification

Designed MIDI-Synthesizer is verified on a C6713 DSP chip which is considered as one the powerful Texas Instrument processors. The CPU is based on a advanced very-long-instruction-word (VLIW) architecture which makes it a good target to implement our synthesizer due to its multifunction property. This processor consists of eight independent functional units including six arithmetic logic units (ALUs), two multipliers and operates at 225 MHz clock rate. DSK board provides a 32-bit AIC23 stereo codec with four 3.5 mm audio jacks (two inputs and two outputs), 16 Mbytes of synchronous DRAM and 512 Kbytes of non-volatile Flash memory which all make it a perfect choice for implementation of audio applications [7].

To realize the design of synthesizer, Code Composer Studio as a useful integrated development environment (IDE) included real-time analysis capabilities, debugger, C/C++ Compiler, Assembler and Linker provides easy to use software to build and debug our programs.

Developed code comprises a MIDI decoder to decode MIDI events received by host computer and 8 channels to play that among 6 channels are polyphonic. All these channels except one which is used for drum have employed FM synthesis to generate sound. To test the synthesizer MIDI-file of the famous song ‘I will survive’ imported to DSK board. The output sound with sampling frequency of 8KHz was rather acceptable. With the current code the CPU usage is %65 at peak which is very promising thanks to FastRTS library to implement cosine function efficiently.

V. Hardware Approach

From production perspective, designing a hardware on ASICs has the benefits of low cost in high volume and ease of use as a module in a system on a chip. According to different applications, variety of strategies may be used in designing ASICs leading to either minimum area or maximum speed. Here we have developed a typical approach from hardware aspect to MIDI-synthesizers.

Fig. 5 shows the top block diagram of the synthesizer which may be implemented on the hardware. The FM generators produce the frequency modulated signals by using the values on the input ports simultaneously. Every channel in the MIDI file is implemented by one FM block. The outputs of all FM blocks accumulate to produce an audio sample. If other methods were used for synthesis, they may be added to parallel blocks. Decoder section extracts the track data from the input MIDI file and provides the input variables for all parallel blocks in every sample. It also keeps track of the time instants in which the musical notes should be played.

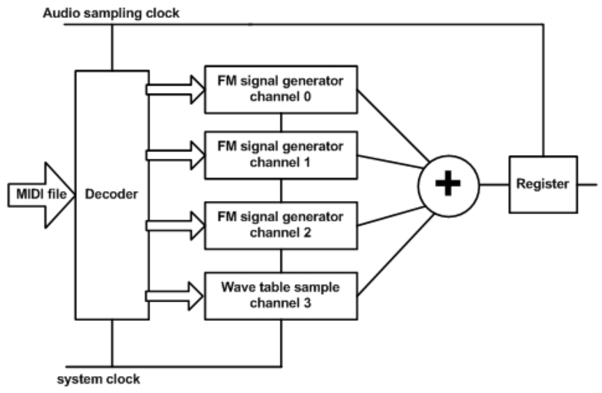
Audio sampling clock
FM signal generator channel 0
FM signal generator channel 1
FM signal generator channel 2
Wave table sample channel 3
MIDI file
Decoder
+
Register
system clock

Fig. 5. Top-level block diagram of suggested MIDI-Synthesizer

Fig. 6 indicates the folded version of the previous system. Here, the FM blocks are implemented in a time multiplexed hardware. Instead of having parallel FM blocks, the same hardware is used for all blocks. As the required rate for generation of samples is equal to the audio sample period, processing time is not a critical issue here. As shown, the FM block works with a considerably higher frequency in comparison with audio sampling frequency.

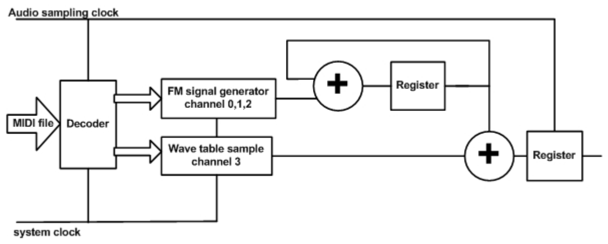

Fig. 6. Time-multiplexed block diagram of suggested synthesizer

As explained through equation (1) the main function to produce the FM signal is the cosine function. Fig. 7 depicts the minimum required blocks to implement FM block. One may use three or more cosine blocks to increase control level on FM output. The cosine blocks work at system frequency and may be realized with different architectures.

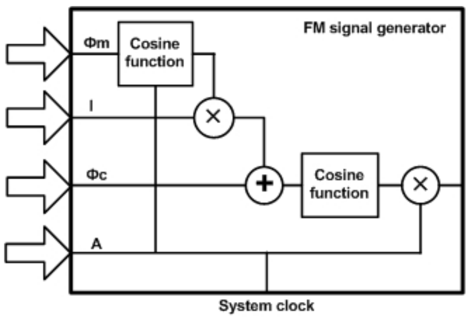

Fig. 7. Internal functions of FM block

Here we have shown the use of CORDIC algorithm to realize a cosine function [8]. An iterative CORDIC architecture based on bit parallel design is illustrated in Fig. 8. The duration of calculation depends on the levels of calculations i.e. precision. For n-level CORDIC process we need at least n clock cycles. We also need a ROM to store the required 'arctan' values as a look-up table. On top of this architecture, a state machine controls the operation and selects the degree of shift and ROM addresses in each iteration. Using a bit serial design in which the input values are being entered in a serial manner may save some more area [9]. For the presented configuration in Fig. 8 , if we can achieve system frequency of F_s and considering two cosine functions for each FM block, the processing rate for each FM clock is approximately F_s/n and for m blocks it is equal to $F_s/(m \times n)$. As an example with system frequency of 100Mhz, 10 FM blocks and 40 level cosine functions, we roughly can reach 250KHz sample rate for audio. According to application we can forfeit sampling rate and get a smaller design.

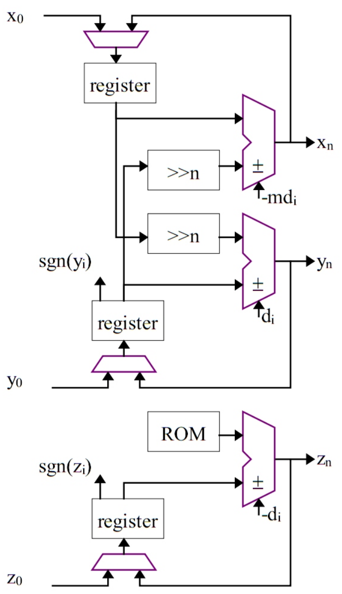

Fig. 8. Bit parallel iterative architecture of CORDIC structure [9]

VI. Conclusion

In this paper, design and implementation of MIDI-Synthesizer are explained from hardware and software perspective. Based on simulated sound of piano, brass and bell in MATLAB, algorithm for decoding MIDI-file and synthesis of eight 4-polyphony channels has been proposed. MIDI-Synthesizer is realized using DSK6713 hand optimized C language which makes the architecture to be capable of implementing more polyphony channels. Discussion on hardware implementation provides a reference model for ASIC design with usage in small and portable digital synthesis systems.

Acknowledgment

The authors would like to thank Martin Stridh and Frida Sandberg for their valuable helps and Bengt Mandersson for his precious suggestions.

References

[1] M. Chunjing, G. Yong and L. Yongmei, *Research and Design of Digital Synthesizer Based on MATLAB*, IEEE/ASME Int. Conf. Mechtronic, Embedded Systems, Applications. , pp. 333-336, 2008
[2] J. Rothstein, *MIDI - A Comprehensive Introduction*, A-R Editions, 1992.
[3] C. Roads, *The computer music tutorial*, MIT Press, 2004
[4] D. McCarron and M. Feibus, *PC Audio Technology and Chip Sets*, Mercury Research, 1994
[5] C. Dodge and T. Jerse, *Computer Music*, Schirmer Books: New York, NY, 1985
[6] M. Russ, *Sound Synthesis and Sampling*, Elsevier, 2004
[7] TMS320C6x Users Guide, *Digital Signal Processing Solution*, 1999
[8] R. G. Harber, J. Li, X. Hu, S. C. Bass, *Bit-Serial Cordic circuits for use in a VLSI silicon compiler*, IEEE Int. Symp. Circuits and Systems, vol 1, pp.154-157, 1989
[9] R. Andraka, *A survey of CORDIC algorithms for FPGA based computers*, citeseer, 2001