C-BASED HARDWARE DESIGN OF IMDCT ACCELERATOR FOR OGG VORBIS DECODER

Shinichi Maeta[1], *Atsushi Kosaka*[1], *Akihisa Yamada*[1, 2], *Takao Onoye*[1], *Tohru Chiba*[1, 2], *and Isao Shirakawa*[1]

[1]Department of Information Systems Engineering,
Graduate School of Information Science and Technology,
Osaka University
2-1 Yamada-oka, Suita, Osaka, 565-0871 Japan
phone: +81 6 6879 7808, fax: +81 6 6875 5902,
email: {maeta, kosaka, onoye, sirakawa}@ist.osaka-u.ac.jp

[2]Sharp Corporation
2613-1 Ichinomoto, Tenri, Nara, 632-8567 Japan
phone: +81 743 65 2531, fax: +81 743 65 3963,
email: yamada.akihisa@sharp.co.jp,
chiba@slab.tnr.sharp.co.jp

ABSTRACT

This paper presents hardware design of an IMDCT accelerator for an Ogg Vorbis decoder using a C-based design system. Low power implementation of audio codec is important in order to achieve long battery life of portable audio devices. Through the computational cost analysis of the whole decoding process, it is found that Ogg Vorbis requires higher operation frequency of an embedded processor than MPEG Audio. In order to reduce the CPU load, an accelerator is designed as specific hardware for IMDCT, which is detected as the most computation-intensive functional block. Real-time decoding of Ogg Vorbis is achieved with the accelerator and an embedded processor both run at 36MHz. The operation frequency is at the same level as that of MPEG Audio decoding process by an embedded processor.

1. INTRODUCTION

With recent advances in audio compression technology, it has seen a steady increase in the popularity of digital music that is delivered over the Internet. Therefore, there are strong demands for portable audio players supporting various audio compression formats.

Ogg Vorbis [1] is a new audio compression format, which is distinctive in terms of fully open, non-proprietary, and patent-and-royalty-free. The original Ogg Vorbis reference decoder is performed with floating-point arithmetic, which makes it difficult to achieve realtime decoding by an embedded processor with no FPU (Floating Point Unit). A fixed-point arithmetic version of the Ogg Vorbis decoder, called Tremor [1], was released in September 2002, which enabled an embedded processor to decode Ogg Vorbis bitstream in realtime. However, its decoding process still requires higher computational cost than that of MPEG Audio [2, 3], and suffers from high power consumption.

Motivated by this technical issue, the present paper proposes hardware design for an Ogg Vorbis audio decoder with an embedded processor. In order to reduce CPU load, a functional block with high computational cost in Ogg Vorbis decoding, specifically IMDCT, is implemented by dedicated circuits through the use of a C-based design system. An ARM7TDMI is used as the embedded processor since it has come into wide use recently.

2. OGG VORBIS CODEC

2.1 Ogg Vorbis Overview

Figure 1 shows a block diagram of the Ogg Vorbis codec processes outlined below.

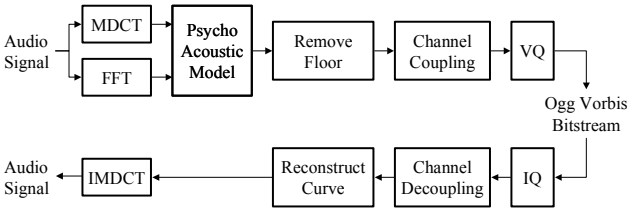

Figure 1: Process Flow of Ogg Vorbis.

- MDCT, FFT:
  Coefficients in the time domain are transformed to those in the frequency domain by MDCT and FFT, where MDCT coefficients and FFT coefficients are used as encoding data and parameters in the succeeding modules, respectively.
- Psycho Acoustic Model:
  Inaudible spectrums are removed on the basis of "psycho-acoustic" masking and a silence threshold.
- Remove Floor:
  A spectrum envelope, called a floor curve, is generated. Then, MDCT coefficients are divided by their floor curve values to flatten the shape of spectrums.
- Channel Coupling:
  Further data compaction is attempted by correlating the two channels.
- Vector Quantization:
  A set of spectrums, which represents a particular region, is approximated by a codevector.

To the contrary, the decoding algorithm is accomplished by tracing above mentioned processes in the reverse order, as illustrated in Figure 1.

2.2 Software Simulation

In order to estimate the operation frequency required for realtime decoding, the execution cycle of the Tremor decoding process was analyzed. In this analysis, the ARM7TDMI was used as the target processor and the decoding process was executed with sample bitstreams of a *violin*, *trumpet*, and *flute* encoded by the Ogg Vorbis Version 1 encoder at the sampling rate of 44.1kHz and the target bitrate of 128kbps. To generate the profile summary of the decoding process, each sample was decoded by using ARMulator (an emulator of ARM processors) with zero memory wait. Table 1 summarizes the result of profiling.

Table 1: Profile Summary – Num. of Cycles

	cycle	num. of frame	cycle/frame
violin	4,095,509,129	2,592	1,578,839
trumpet	2,030,970,723	1,540	1,318,812
flute	3,730,821,502	3,015	1,237,420

As can be seen from this table, the number of cycles needed for 1 frame is calculated as 1,578,839. Since the allowable time for processing a long frame (1,024 samples) at 44.1kHz is 1,024/44,100 = 23.2[ms], the required operation frequency of the embedded processor is determined as 1,578,839/23.2 = 68.1[MHz]. As reported in [2] and [3], other audio decoding processes such as MP3 and MPEG-2 AAC are executed generally around 30MHz of operation frequency by using ARM7TDMI. Contrary to this, Tremor decoding process requires almost twice of the clock frequency. Thus it should be pointed out that the operation frequency must be reduced by employing an accelerator for a functional block with high computational complexity.

In order to evaluate the computational complexity in detail, computational cost for each functional block in the Tremor decoding is analyzed as is summarized in Table 2.

Table 2: Profile Summary - Execution Time

	IMDCT	IQ	Others
violin	51.12%	11.12%	37.76%
trumpet	51.33%	11.21%	37.46%
flute	51.12%	11.12%	37.76%

According to the result of this analysis, the IMDCT process takes more than half of the total processing time. Therefore, our Ogg Vorbis audio decoder is designed to be equipped with IMDCT accelerator.

3. C-BASED DESIGN OF IMDCT ACCELERATOR

3.1 Design Method

In contrast with conventional RTL-based design, several C-based design systems have been developed to make higher level design more efficient. The Bach system is a C-based design system which has been applied to several real LSI designs [4]. The user language of the Bach system, called Bach C, is designed based on ANSI-C. Since basic grammar of Bach C is the same as that of ANSI-C except several extensions, designers can write Bach C code in the same way as the conventional C code. Therefore designers can utilize existing software library in ANSI-C for hardware designs. The Bach system consists of mainly a synthesizer and a simulator. The synthesizer includes high level synthesis technique and generates register-transfer level circuits in VHDL. The simulator translates Bach C code into ANSI-C and compiles it using a conventional C compiler. Since the simulator runs more than 100 times as fast as RTL simulators, design validation period can be drastically reduced.

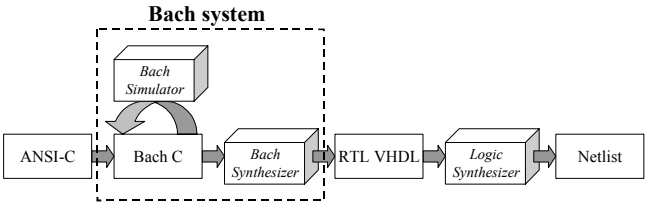

Figure 2:Design Flow using Bach System.

Figure 2 shows the design flow. First we modify the original ANSI-C code so that it can be synthesized by the Bach synthesizer. What we need to do at this phase are:

1) to specify bit width for each variable if necessary, and
2) to rewrite a dereference operator (*) of specifying an array element to an operator [].

We improve the Bach C code while validating it using the Bach simulator. Once register-transfer level VHDL code is generated, the code can be easily synthesized into netlist by conventional logic synthesizer.

3.2 Hardware Implementation of IMDCT

When we design hardware, we often need to consider different cost from that of software implementation. In software implementation, it is important to minimize the total number of operations since the size of hardware (processor/RAM/ROM) is fixed. In hardware implementation, it is important to take account of trade-off between area and operation cycles.

Suppose the following two functions $f1$ and $f2$.

$$f1: x, y, z \rightarrow x \times y + z;$$
$$f2: x, y \rightarrow x \times y;$$

For software implementation, it would be better to have two functions $f1$ and $f2$, although $f2$ can be replaced with $f1$ by setting z to 0. For hardware implementation, it would be better to have a module for only $f1$ than to have two modules for $f1$ and $f2$ in spite of increasing the total number of operations.

High level synthesis in the synthesizer tries to minimize area by sharing functional units among operations in input code, where the sharing is performed to operation by operation. Therefore the synthesizer often fails to share a module, a set of functional units, among several routines in more

complicated cases. Hence, it is important to write a common routine used as many times as possible in input source code without degrading the original performance.

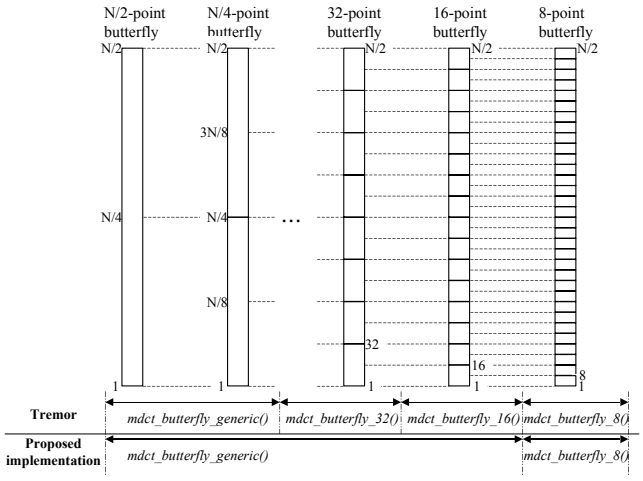

Figure 3: Recursive execution of butterfly calculation.

Tremor adopts Sporer's algorithm [5], whose features are low computational cost based on recursive execution of butterfly calculations for IMDCT. As shown in Figure 3, *N*-point ($N = 2^n$, n is a natural number) MDCT coefficients are transformed into *N*/2-point DCT coefficients, and then DCT is performed through recursive execution of butterfly operations. Tremor recursively executes this process down to 8-point butterfly calculation. At each butterfly calculation, the following transformation is applied as is illustrated in Figure 4.

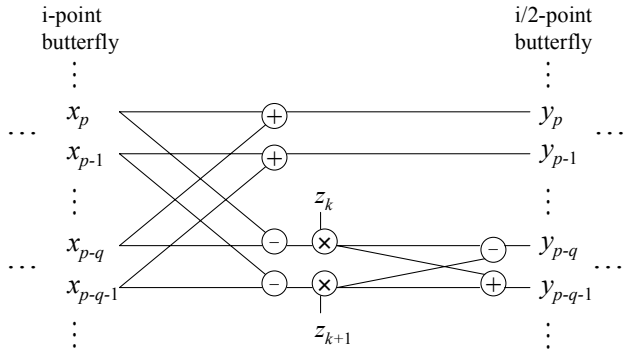

Figure 4: Basic butterfly calculation.

$$
\begin{aligned}
y_p &= x_p + x_{p-q} \\
y_{p-1} &= x_{p-1} + x_{p-q-1} \\
y_{p-q} &= (x_p - x_{p-q}) z_{\frac{n}{2}-m} - (x_{p-1} - x_{p-q-1}) z_{\frac{n}{2}-m+1} \\
y_{p-q-1} &= (x_p - x_{p-q}) z_{\frac{n}{2}-m} + (x_{p-1} - x_{p-q-1}) z_{\frac{n}{2}-m+1}
\end{aligned}
\qquad (1)
$$

$$
q = \frac{N}{2^{l+2}},\ m = 2^{l+3},\ z_k = \cos(\tfrac{2k\pi}{N}),\ z_{k+1} = -\sin(\tfrac{2k\pi}{N})
$$

$$
\text{for } l = 0,\ldots,n-4
$$

```
for(i=0;i<M;i++){
  mdct_butterfly_generic();
}

for(i=0;i<N/2;i+=32){
  mdct_butterfly_32(){
    …
    mdct_butterfly_16(){
      …
      mdct_butterfly_8();
    }
  }
}
```

(a) Tremor

```
for(i=0;i<M+2;i++){
  mdct_butterfly_generic();
}

for(i=0;i<N/2;i+=8){
  mdct_butterfly_8();
}
```

(b) Proposed implementation

Figure 5: Implementation Example.

The rough code of Tremor is shown in Figure 5(a). *mdct_butterfly_generic()* function is used to execute butterfly calculation down to 64 points recursively. *mdct_butterfly_32()* and *mdct_butterfly_16()* functions are used for 32 and 16-point butterfly calculation, respectively, and finally, *mdct_butterfly_8()* function is used for 8-point butterfly calculation. These dedicated functions are implemented for reducing the number of additions/multiplications, since some MAC (multiply-accumulation) operations can be omitted in the calculations.

To implement Tremor as hardware, it is important to use a single function for as many calculations as possible, as mentioned above. According to Sporer's algorithm [5], all butterfly calculations can be executed in a single function regardless of the number of point. Here, *mdct_butterfly_generic()* has four loops corresponding to four quadrant and each loop calculates in Equation (1). It is easily confirmed that this function can be used for butterfly calculations for 16 points or more. Thus we use *mdct_butterfly_generic()* for 16 and 32-point butterfly calculations by rewriting slightly. As for 8-point, we use a dedicated function *mdct_butterfly_8(),* since it can be implemented as small hardware without any MAC. As a result, in the proposed implementation, all butterfly calculations can be executed by only two functions, *mdct_butterfly_generic()* and *mdct_butterfly_8()* as is shown in Figure 5(b).

4. IMPLEMENTAION RESULT

IMDCT accelerators based on the proposed implementation and Tremor have been implemented through the use of Bach C, and compiled using the Bach synthesizer at clock rate of 10MHz. The design period starting from the original code of Tremor is about one week. Table 3 shows the implementation results, where the number of gates was estimated through the use of Virtual Silicon Technology 0.18μm CMOS library as the target library of the Bach synthesizer. The number of execution cycles was estimated by performing a 2,048-point IMDCT at RTL simulation using Mentor Graphics *ModelSim*.

Table 3: Gate Sizes and Execution Cycles

	num. of gates	cycle/frame
Tremor	118,689	42,939
Proposed implementation	87,872	43,432

As shown in Table 3, about 26% reduction on the number of gates is achieved by the proposed approach.

Table 4 shows the numbers of additions and multiplications for a 2,048-point IMDCT in Sporer's algorithm [5], Tremor, and the proposed implementation.

Table 4: Number of Operations

	addition	multiplication
Sporer's algorithm [5]	27,648	15,360
Tremor	15,744	10,880
Proposed implementation	17,408	12,288

The numbers of additions and multiplications increase about 9% and 11%, respectively, compared to that of Tremor. However, the number of execution cycles increases about 1% as shown in Table 3. We can conclude that the proposed implementation is suitable for hardware implementation according to the trade-off between circuit area and performance.

To determine the required operation frequency of ARM7TDMI and the proposed IMDCT accelerator, the number of gates and the number of execution cycles required for the proposed IMDCT accelerator are estimated by using Bach synthesizer at 10MHz, 20MHz, 30MHz, and 40MHz. Table 5 summarizes the required operation frequency of ARM7TDMI along with the variations of the proposed IMDCT accelerator. The number of gates and the number of execution cycles are estimated in the same manner as Table 3.

Table 5: Required Operation Frequency

IMDCT Accelerator	num. of gates	cycle/frame	ARM7TDMI
10MHz	87,872	43,432	40.9MHz
20MHz	87,684	43,813	36.7MHz
30MHz	**83,482**	**44,835**	**35.6MHz**
40MHz	80,730	45,474	34.9MHz

As a result, 36MHz is the suitable operation frequency for realtime decoding using ARM7TDMI and the IMDCT accelerator. Then the proposed IMDCT accelerator has been implemented through the use of Synopsys *Design Compiler* by the Virtual Silicon Technology 0.18μm CMOS library. Table 6 summarizes the main features of the IMDCT accelerator.

Table 6: IMDCT Accelerator Specification

num. of gates	59,655
cycle/frame	44,835
Maximum clock rate	40.2 MHz
Internal ROM	32-bit×2,050

Consequently, the proposed IMDCT accelerator enables Ogg Vorbis decoding in realtime by using the embedded processor running at the same level of operation frequency for MP3 and MPEG-2 AAC decoding.

5. CONCLUSION

This paper has described hardware design of an IMDCT accelerator for an Ogg Vorbis decoder. We could implement it in very short period by using a C-based design system. Implementation results demonstrated that the operation frequency of an embedded processor could be reduced to 36MHz by using the proposed accelerator. Development is continuing on the extension of our IMDCT approach to handle with both Ogg Vorbis and other audio compression formats, and SoC integration for single chip audio decoder.

REFERENCES

[1] Xiph.Org Foundation, "The Ogg Vorbis CODEC project," http://www.xiph.org/ogg/vorbis/.

[2] K. Lee, Y. Park, and D. Youn, "Software optimization of the MPEG-audio decoder using a 32-bit MCU RISC processor," *IEEE Trans. on Consumer Electronics*, vol. 48, pp. 671–676, Aug. 2002.

[3] ARM Ltd, http://www.arm.com/.

[4] T. Kambe, A. Yamada, K. Nishida, K. Okada, M. Ohnishi, A. Kay, P. Boca, V. Zammit, and T. Nomura, "A C-based synthesis system, Bach, and its application," in *Proc. Asia and South Pacific Design Automation Conference (ASP-DAC) 2001*, pp. 151–155, Jan. 2001.

[5] T. Sporer, K. Brandenburg, and B. Edler, "The use of multirate filter banks for coding of high quality digital audio," in *Proc. 6th European Signal Processing Conference (EUSIPCO)*, Amsterdam, vol. 1, pp. 211–214, June 1992.