A FIXED-POINT MPEG AUDIO PROCESSOR OPERATING AT LOW FREQUENCY

Yongseok Yi and In-Cheol Park
Department of Electrical Engineering and Computer Science, KAIST
373-1, Kusong-dong, Yusong-gu, Taejon 305-701, Korea
E-mail: yslee@ics.kaist.ac.kr, icpark@ee.kaist.ac.kr

Abstract – **A fixed-point pipelined processor optimized for decoding MPEG-1 audio layer III (MP3) is presented. Various examination and experiments on the decoding algorithm are exploited to lower the operating frequency by providing efficient instructions and addressing modes, because low frequency is directly related to low power consumption. Accordingly, the dynamic scaling method is used to preserve the precision while computing in fixed-point arithmetic. Also, a novel instruction set tuned for MP3 decoding is presented. The resulting cycle count required to decode a frame is so small that the proposed processor can operate with 12.8 MHz while decoding in real-time.**

Index Terms – **MP3, low power, fixed-point arithmetic, multiply-accumulate, programmability.**

I. INTRODUCTION

Portable devices equipped with MPEG-1 audio layer III (MP3) decoding capability are becoming popular. In addition to performance and area, the power consumption problem is becoming a critical design factor as applications come to the portable domain.

The first approach to implement the MP3 decoding in hardware was taken by Maturi [1], who presented a single-chip MPEG audio decoder. It could decode layer 1 and 2, and operated at 25-30 MHz. Witte, et al. [2] have presented a multipurpose digital signal processor using a 20-bit fixed-point arithmetic, running at 20 MHz. While [1] and [2] are based on programmable processors, Tsai, et al. [3] used hard-wired control to reduce control overhead. All memory banks are 24-bit wide and the chip is reported to operate at 11.5 MHz. However, the hard-wired control means that there exists no programmability. One of the commercially available decoder is the MAS 3507 chip [4], which is capable of MPEG 1/2 layer 2/3. The chip consists of a 20-bit processor and special purpose circuits working at a 14.725 MHz clock. However, it is doubtful that the core DSP actually operates at the supplied frequency. Rather, its operating frequency might be a multiple of 14.725 MHz. The latest implementation concentrated on realizing the full decoding in a single chip by including an internal digital-to-analog converter and an internal analog-to-digital converter [5]. The decoding core is based on 32-bit floating-point DSP working at 27MHz, which is luxurious for the application, MP3 decoding.

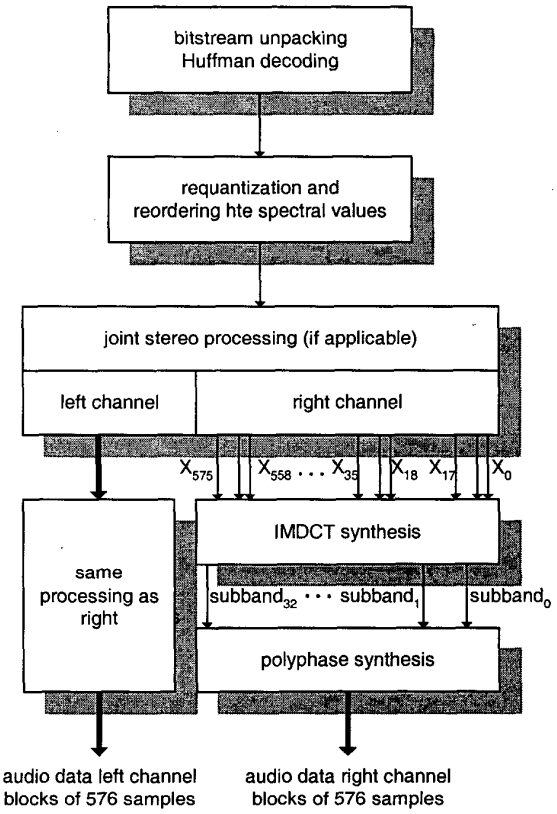


Fig. 1 MP3 decoding flow chart

All the embedded processors or DSP processors employed in the above implementations are developed based on conventional instructions and architectures, thus the processors are not tuned for MP3 decoding, leading to higher operating frequency and power consumption than expected.

This paper mainly focuses on the design of a fixed-point processor optimized for MP3 decoding. By exploiting various examinations and experiments on the MP3 decoding algorithm, which is depicted in Fig.1, a novel instruction set is proposed to reduce the cycle count consumed in the computationally intensive stages. Among the stages, the IMDCT synthesis and the polyphase synthesis are most computationally intensive. The reduction

Contributed Paper
Manuscript received July 16, 2001

of cycle count required to decode a frame is critical in determining the lowest frequency that the decoder can operate at. In algorithmic and architectural view, lowering the operating frequency (f) and/or reducing the effective load capacitance (C_L) of the circuitry achieve low power consumption.

Using simple fixed-point arithmetic instead of floating-point arithmetic reduces the circuit area and lessens the time budget as it leads to the reduction of f and C_L. Reducing the bit-width of memory and registers also helps reduce C_L. Since MP3 decoding is a real-time application, the time allowed for the decoding is fixed. If we reduce the cycle count for a given time duration, f will be reduced again.

The rest of this paper is organized as follows. Section 2 discusses the fixed-point-based implementation of the MP3 decoding algorithm, Section 3 derives the performance requirement and introduces the new instruction set optimized for MP3 decoding, and Section 4 describes the architectural design in detail.

II. FIXED-POINT IMPLEMENTATION

Although a fixed-point implementation can result in an area- and power-efficient architecture working at a lower frequency, it suffers from the risk of precision loss. For the fixed-point implementation, a floating-point version of MP3 decoding program is converted to an integer version, where we have to determine the minimal bit-width required for data processing. The bit-width must be minimized to reduce hardware complexity and power consumption, while preserving the precision required for ISO/IEC 11172-3 compliance. For example, the first precision loss arises in the requantization stage where the Huffman data is requantized to spectral values (see Fig. 1). The equation is,

$$xr_i = \mathrm{sign}(is_i) \times |is_i|^{\frac{4}{3}} \times 2^{\frac{1}{4}(\mathrm{global\_gain}[gr]-210)} \times 2^{-(\mathrm{scalefac\_multiplier} \times (\mathrm{scalefac\_l}[sfb][ch][gr] + \mathrm{preflag}[gr] \times \mathrm{pretab}[sfb]))} \quad (1)$$

In (1), xr_i is the spectral value requantized from the Huffman decoded value is_i. The parameters global_gain, scalefac_multiplier, scalefac_l, preflag and pretab are all decoded from the audio data in MP3 bitstream. As can be seen above, a spectral value, xr_i, is made up of four terms, namely, sign (is_i), $|is_i|^{4/3}$, $2^{0.25(\mathrm{global\_gain}\ [gr]-210)}$ and $2^{-(\mathrm{scalefac\_multiplier}*\mathrm{scalefac\_l}\ [sfb][ch][gr]\ +\mathrm{preflag}\ [gr]*\mathrm{pretab}\ [sfb])}$. The latter two terms are actually pre-computed and stored into tables, rather than directly computed.

To explain an example of precision loss problem, let us denote $2^{0.25(\mathrm{global\_gain}\ [gr]-210)}$ as *globscale*. The possible range of *globscale* is [1.57009245868x10^{-16}, 2435.49617153], where the dynamic resolution is 384 dB. It requires 70 bits to cover the whole dynamic range when represented in fixed-point notation.

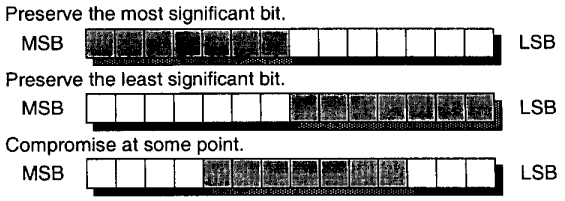


Fig. 2 Choices of scaling

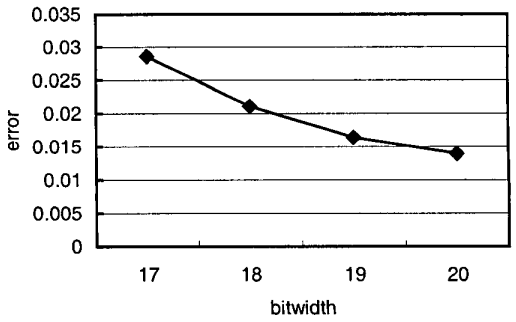


Fig. 3 The RMS error for varying bitwidths

A. Dynamic Scaling

Since we are not allowed to use the bitwidth as wide as required, we have to somehow scale the values to fit in a rather short bitwidth. If the bitwidth is not wide enough to cover the dynamic range, some of the values are lost. The amount of scaling determines which portion of the values would be lost. Two extreme choices are shown in Fig. 2. If the values are scaled to preserve the most significant bit, small values will be lost. On the other hand, if the values are scaled to preserve the least significant bit, large values will be lost. Therefore we need to make a compromise between the two choices.

A granule, half of a frame, consisting of 576 sample data is the basic unit of MP3 decoding. Not all the sample values reach the maximum of their range. Therefore the compromise is made at the point where the scaled values just cover the largest data in the granule. In this manner, the scale factor varies from granule to granule. This factor, determined for a granule, is kept throughout the rest procedures to be used for rescaling the output data at the final stage.

B. Finding minimum bitwidth

Audio applications require only limited bit-width less than conventional 32 bits. To find the minimum bitwidth with minimum precision loss, we ran the C model of MP3 decoding algorithm with fixed-point arithmetic, varying the bitwidths. Fig. 3 depicts the result of the experiment.

The ISO/IEC 11172-3 compliance requires for a provided test sequence that the RMS level of the difference signal between the output of the decoder under test and the supplied reference output is less than 2^{-11}/sqrt(12) for a sine

sweep (20 Hz – 10 kHz) with an amplitude of –20 dB relative to full scale [6]. As can be seen in the figure, the bitwidth should be more than 20 to satisfy the compliance's requirement. The bitwidth of memory, register, and datapath is therefore set to 20.

III. Instruction Set

The minimum performance of a MP3 decoder must guarantee real-time decoding. MP3 supports three sampling frequencies, or, the number of samples per second: 32, 44.1 and 48 kHz. With the fact that one frame contains 1152 samples, we can derive how many frames must be decoded in one second, or how short the decoder must take to decode one frame. When the highest sampling frequency is applied, the decoder should be able to decode one frame in 1152/48000 = 0.024 seconds. Note that the performance requirement is independent of the bit rate.

Once the time constraint is given, we need the target clock frequency. By the low-power strategy, we have first targeted 13.5 MHz, which is the system clock frequency of MPEG-1. Given this frequency the MOPS value is calculated as below.

$$\begin{aligned}&(\text{clock frequency})\times(\text{second per frame})\\&=(13.5\times10^6)\times(0.024)\\&=0.324\ \text{MOPS}\end{aligned}\qquad(2)$$

The instruction set of our audio processor was developed on the basis of the implementation of MP3 decoding algorithm using a *basic instruction set*. The basic instruction set refers to the set of instructions that might be supported by almost any architecture.

When described using only the basic instruction set, the decoding algorithm's cycle count for a frame was 1,257,701 cycles, about 4 times the required 324,000 cycles. When all loops were unrolled, the instructions counted 563,789. This is the number of program memory accesses per one frame.

Based on the analysis of the assembly code, the instruction set is modified. First, it was found that NOT (1's complement), XOR (logical exclusive-OR), TST (test) and SHR (shift right logically) were not used at all. These are removed from the instruction set. Moreover, the use of NEG (2's complement) instruction is restricted only to the case in which the operand and result is the same register or memory address.

Second, the sequence [LDR (load a register from memory) → data processing instruction → STR (store a register in memory)] was found very often, i.e. 152223 times. The sequence includes [LDR→STR], which implies a move. This pattern is very natural because multimedia applications, such as MP3, are data-intensive. So rather than just relying on LDR and STR for an operand fetch, letting the data processing instructions fetch their operands directly from memory will remove the overhead of load and store instructions. According to this, all data processing instructions are enabled to fetch operands from memory. This makes LDR and STR of no use because MOV takes their place.

Third, the IMDCT stage and the polyphase synthesis stage include repetitive multiply-and-accumulate operations. This is because the MP3 decoding needs many linear convolution operations. As it will be explained later again, the execution stage of a MAC operation takes two cycles in this architecture. And because a branch wastes two pipeline slots when taken, such operations become the primary cause of slowing the algorithm. If an instruction that can execute many MAC operations is introduced, we could save the branch overhead. So we add the SMAC (successive multiply-and-accumulate) instruction to the instruction set. Moreover, in the proposed architecture, successive MAC's can be pipelined when no other instruction intervenes among them. The SMAC instruction is able to execute up to 32 successive MAC's. It can use the operands only from memory, assuming the operands' addresses are supplied by AGUs.

Fourth, the algorithm inherently contains many loops. To avoid the overhead of software looping, an RPT (Repeat) instruction is added. This instruction repeats a given number of instructions below it for a given number of times. Since the maximum iteration number of the loop is 576, it can repeat the instructions up to 2^{10} = 1024 times, where $10=\lceil \log_2 576 \rceil$. And since the size of the most inner loop body does not exceeds 20 lines, the instruction is made to repeat up to 32 instructions. It does not accommodate nested loops to avoid the increase of hardware complexity.

TABLE 1
Frequently Appearing Instruction Patterns

Instruction pattern	# of appearance
ADD r0, r1, r3 LDR r2, <address>	876
ADD r0, r1, r3 STR r2, <address>	886
NEG r0, r0 LDR r1, <address>	288
SUB r0, r1, r2 STR r3, <address>	576
MOV r0, r1 STR r2, <address>	576
ADD r1, r2, r3 MUL r4, r5, r6	600

Fifth, parallel instructions are exploited. The design of parallel instructions is based on frequent instruction patterns appearing in the assembly code. Of course, the patterns are limited to independent instructions only. These are listed in Table 1.

The length of an instruction is fixed to 20 bits, which is the minimum to hold the whole memory address at once in an

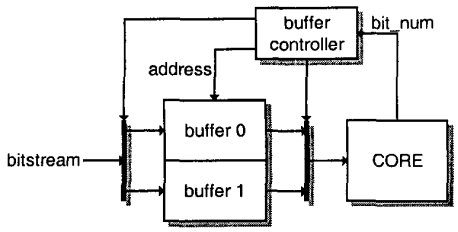


Fig. 4 The environment of the decoding core (bitstream buffers)

instruction. A shorter instruction length cannot accommodate the memory address and then two instructions may have to be used, leading to a severe loss of performance. As the instruction length is too short for an ordinary MOV instruction to contain a memory address, which takes 15 bits, a special instruction is added to load the address registers (AR). It is SETAR to load ARs with a 15-bit immediate or a register value. Other instructions access the memory by indicating ARs.

IV. ARCHITECTURAL DESIGN

The architecture presented in this paper is designed so as to accommodate the instruction set introduced in section III. It is fully pipelined and has two separate memory banks and an additional bitstream buffer to hold the incoming MP3 bitstream. The bitstream buffer is a dual buffer, as shown in Fig. 4, that when the core accesses buffer 0, the input bitstream is stored in buffer 1 and vice versa. Therefore we assume that the core can decode the bitstream continuously without having to wait for the buffer to be filled.

A. *Pipeline stages*

As discussed above, almost instructions are able to fetch their operands from data memory. The conventional 5-stage pipeline commonly used in general purpose RISCs, namely instruction fetch → instruction decode → execution → memory → write back, is not appropriate as the memory access will not arise only in the memory stage. Instead, a DSP-like pipeline was adopted as shown in Table 2. Each stage is isolated by flip-flops using a single-phase clock.

Since not all the instructions' execution stages take a single cycle, the cycle counts of the execution stage may vary with respect to the instructions, e.g. SMAC's iteration number can be up to 32 and its execution stage extends to 33 cycles. Table 2 shows the abstract description of the operations performed in each pipeline stage. There are two stages where memory accesses take place, implying the possibility of structural hazard. If such a structural hazard occurs, there is no choice but to insert a pipeline bubble in the latter instruction to avoid the collision. If this situation occurs often, it will result in severe performance degradation. Fortunately, it was found that such cases rarely existed in the program.

B. *Datapath*

Fig. 5 depicts the datapath along with other non-datapath elements. Detailed control signals are omitted to ease reading. Many of the signal lines going back and forth are the forwarding paths needed for fluent pipeline operation.

As can be seen in the figure, there are 5 operand registers on the border between OF stage and EX stage. The leftmost one, named mov_op, is a register for a move from memory to register files parallel with ADD or SUB instructions.

The loop buffer in IF stage is used when the RPT instruction is executed. In the first iteration, the instructions in program memory are copied into the loop buffer when they are executed. From the next iteration to the end, the loop buffer is accessed instead of the program memory.

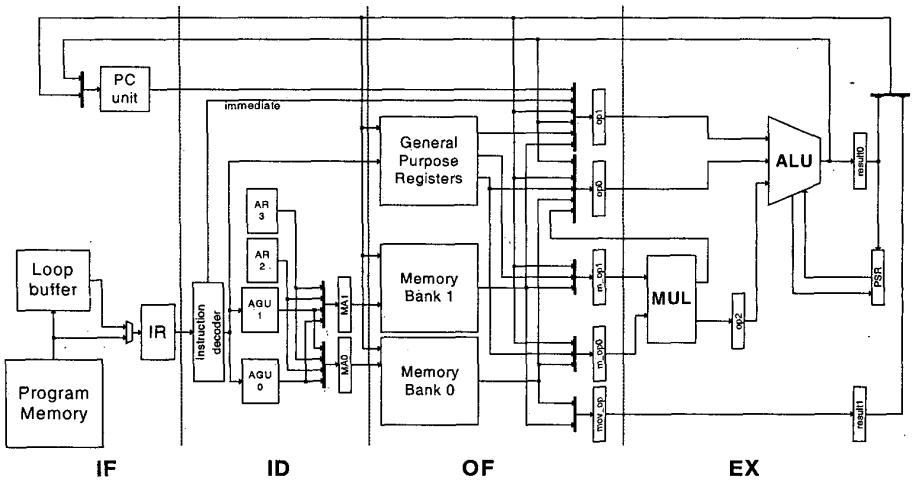
The execution stage is shown in more detail in Fig. 7. The 20-bit x 20-bit multiplier that can execute the multiplication in a single cycle is an array multiplier based on radix-4 Booth's algorithm. The multiplier uses 4-to-2 compressors to sum up the partial products. We do not include the final adder in the multiplier to save area. Instead, the final addition is accomplished by a CLA (carry look-ahead adder) in ALU. The multiplier receives two 20-bit operands and emits a 40-bit sum vector and a 40-bit carry vector.

TABLE 2
OPERATIONS OF PIPELINED STAGES

Stage	Operation
IF	IR ← Program memory or loop buffer
ID	Instruction decoding MA ← AGU output
OF	Operand registers ← Memory or register bank
EX	Multiply operation and/or ALU operation Register bank ← ALU output Result register ← ALU output
MEM	Memory ← Result register

These are held in the registers, op1 and op2, respectively. In the next cycle, they are added together in the ALU to produce the final result. Since the summing of the partial products and the final addition are separated, the multiplication takes two cycles even though it can be done in one cycle, relieving the time budget of the circuit. MAC also consumes two cycles.

Since the EX stage can perform a multiplication and an addition in parallel, the SMAC instruction can be fully pipelined within EX stage. The first multiplication takes place at the multiplier (denoted MUL), and then the sum vector and the carry vector are passed on to the adder through register for the addition in the next cycle. While this addition is being done, the multiplier carries out the next multiplication. From the second addition, the result of the former calculation is fed back as another operand of addition. Fig. 6 and Fig. 7 explains this concept.



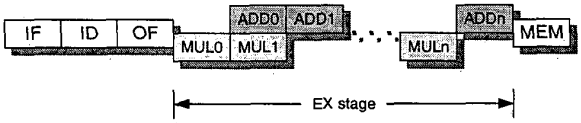


Fig. 6 Pipelined execution of SMAC

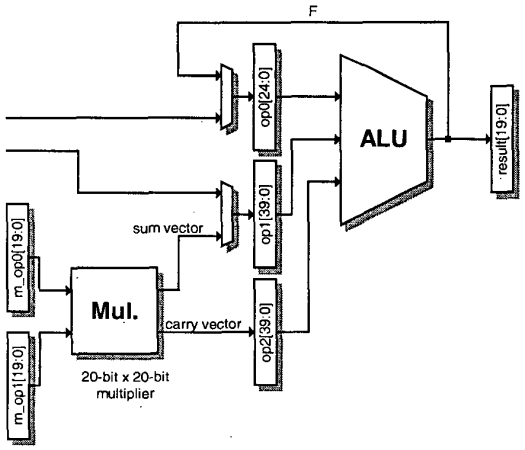


Fig. 7 Block diagram of execution stage

Fig. 8 shows the adder in detail. The adder adds three operands using 3-to-2 compressors. Since the algorithm does not require all the 40 bits of the multiplication result, only the upper 20 bits are calculated. That is, we do not have to use the area-wasting full 40-bit carry-look-ahead adder. However, when only the upper half of the result is computed, there arises an accuracy problem since we have to take the carry propagated from the lower half into account. Thus a 20-bit CG (carry generator) for the lower half is used to keep the result accurately enough. It uses the

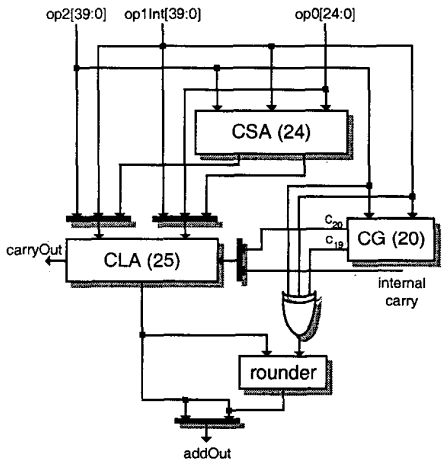


Fig. 8 Block diagram of adder

carry-look-ahead mechanism to generate the carry-in to the upper half.

The 24-bit CSA (carry save adder) is used only for the MAC operation. In the other cases, e.g. add or sub, the operands are directly fed to the 25-bit CLA. The CLA is widened by 5 extension bits, so that at least 32 repetitive MAC operations can be performed without causing overflow. The number '32' comes from the fact that the accumulating loops do not iterate more than 32 times in the MP3 decoding algorithm.

The rounder is used to correct the multiplication result where the lower half is truncated away. If it is just truncated, the results will suffer from round-off errors and they will accumulate as the algorithm proceeds. In this architecture, we adopted round-to-nearest-even scheme [7]. Instead of implementing it using a ROM, a few logic gates

are used because the rounding range is limited to $l = 3$, where l represents the number of input address lines.
Table 3 gives the operation of the rounding scheme. This is a simple combinational logic with 3 primary inputs and 2 primary outputs. The two digits at the left of binary point in the input number are the two least significant bits of the CLA's output, and the leftmost digit corresponds to the 20^{th} bit of the final result of multiplication calculated by the XOR gate in Fig. 8.

TABLE 3
ROUNDER OPERATION

Number	ROM(x)
X00.0	X00.
X00.1	X01.
X01.0	X01.
X01.1	X10.
X10.0	X10.
X10.1	X11.
X11.0	X11.
X11.1	X11.

According to Table 3, the rounder's logic is:

$$\begin{aligned} &\text{roundOut}[1]=\text{claOut}[1]+(\text{claOut}[1]\cdot\text{result}[20]) \\ &\text{roundOut}[0]=(\text{claOut}[1]\oplus\text{result}[20]) \\ &\quad +(\text{claOut}[1]\cdot\text{claOut}[0]) \end{aligned} \tag{3}$$

where roundOut is the output of the rounder, claOut the output of the CLA and result [20] the most significant bit of the lower half.

C. *Address generation unit*

The presented audio processor basically supports register-indexed addressing as well as direct addressing. The address generation unit (AGU) calculates the data memory address according to the specified addressing modes. Fig. 9 and Fig. 10 depict the schematics of AGU0 and AGU1, respectively.
In an MP3 decoder, like many other DSP's, the necessity of modulo addressing arises when the algorithm repeats to refer a set of coefficients, usually stored as arrays, repetitively. This is accomplished by AGU0. The length register in Fig. 9 contains the length of a coefficient block plus the base address value, which is the absolute address of the block's boundary. The value in the length register is compared to the incremented address using a subtractor. If the result of the subtraction is greater than or equal to zero, that is, if the incremented address is greater than or equal to the boundary, the initial value of base address, which had been copied to AR0_copy, is selected as output. AR0 is also restored to its initial value at the same time.
AS0, the address stride register, contains a value by which AR0 is incremented. AM0, the address mode register, and AS0 can be directly accessed by software.

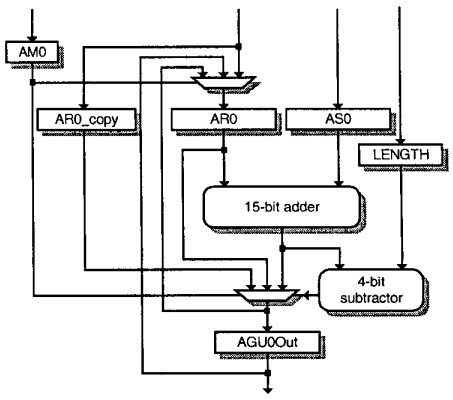


Fig. 9 The schematic of AGU0

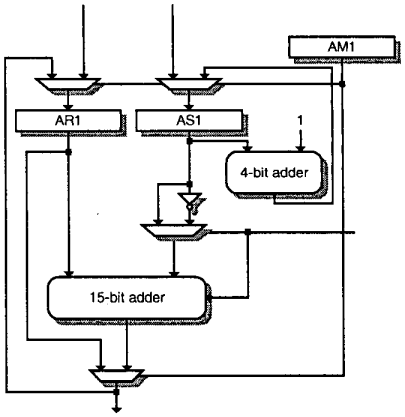


Fig. 10 The schematic of AGU1

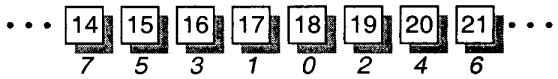


Fig. 11 Partially symmetric address pattern

AS1	Operation	AR1
0	+	N
1	–	N – 1
2	+	N + 1
3	–	N – 2
4	+	N + 2
:	:	:

Fig. 12 Computation of partially symmetric address

In the alias reduction stage of MP3 decoding algorithm, a symmetrically increasing and decreasing address pattern is observed, as depicted in Fig. 11. The numbers in the boxes represent addresses and the numbers below them represent the order of accesses. This partially symmetric addressing is performed by AGU1. Partially symmetric address requires

subtractions from the base address. So the AGU is designed to perform the subtraction using an adder. Controlled by the control unit, the adder takes AS1 inverted and the carry in as 1 when subtraction is needed. The subtraction and addition mode is switched cycle by cycle. Along with this, AS1 is incremented using another 4-bit adder and the address register AR1 keeps being updated. Fig. 12 illustrates the computation assuming that the initial value of the address sequence is N.

V. Results and Comparisons

Fig 13 shows the RMS error of requantized values using different scaling methods enumerated in Table 1. As can be seen in the figure, dynamic scaling guarantees the maximum precision above all.

The number of cycles to decode one frame is reduced due to the new instruction set. This is reported in Table 4 and Fig. 14.

As expected, most of the cycles reduced are due to the SMAC instruction. The amount of reduction due to RPT is small compared to SMAC's contribution, because the number of eliminated instructions is just 4 × (# of iterations) for an ordinary loops, while an SMAC can substitute 10 x (# of iterations) at most. However, the instruction significantly reduces the number of instruction memory accesses that consume much power.

With the final cycle counts in Table 4, we can estimate the operating frequency. The estimated clock frequency is

$$\frac{306139 \text{ cycles}}{0.024 \text{ seconds}} = 12.8 \text{ MHz}$$

The result is lower than the target frequency by about 5.2 %. In Table 5, the proposed decoder is compared with previous implementations.

The bit-width of the proposed decoder is equal to the previous minimum, and the resulting operating frequency is lower than any previous implementations but [3] which has no programmability at all.

TABLE 5
RESULT COMPARISON

	Freq. (MHz)	Bitwidth (bits)	Flexibility	Arithmetic
[1]	25-30	24	High	N/S*
[2]	20	20	High	Fixed-point
[3]	11.5	24	Low	N/S
[4]	14.73	20	High	N/S
[5]	27	32	High	Float-point
Proposed	12.8	20	High	Fixed-point

* N/S: Not specified

A fully synthesizable Verilog HDL model supporting the whole instruction set is described in RTL level to implement the proposed chip. The gate count of the synthesized circuit is 25,736. Fig. 15 shows the layout of the circuit, which is place-and-routed using 0.35-μm standard CMOS technology. It has two working memory banks and a

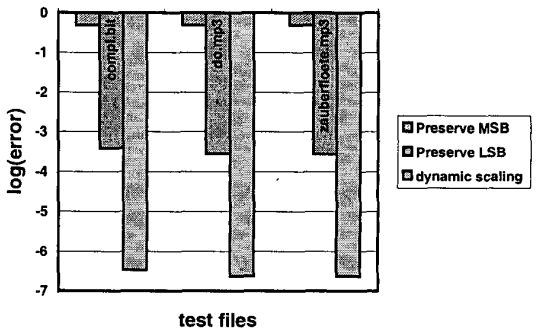


Fig. 13 Comparison of scaling methods

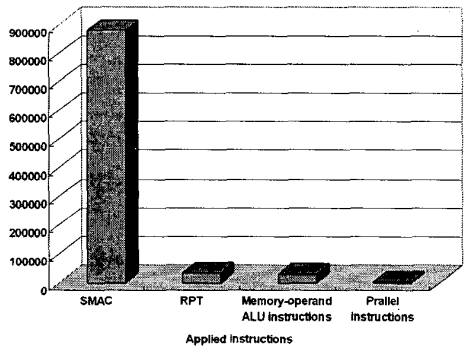


Fig. 14 Cycle count reduction

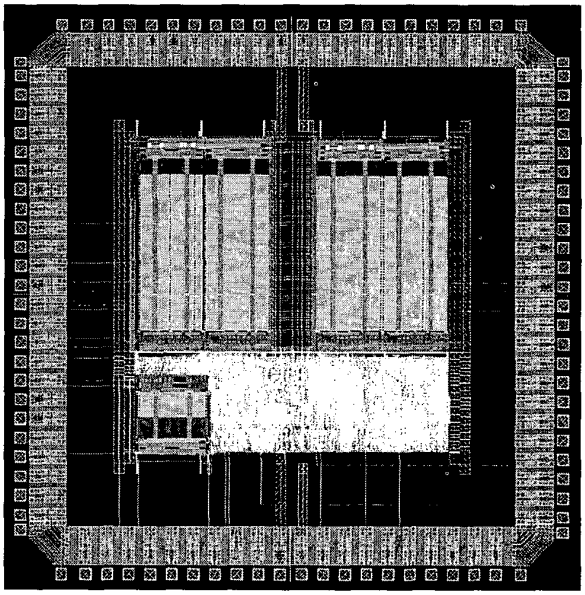


Fig. 15 Layout of the audio processor chip

bitstream buffer fed by a serial input. The size of the core, including memory blocks, is 2.4 mm x 2.4 mm and the size of the die is about 4 mm x 4 mm. The chip is supplied by the power of 3.3 V. The digital-to-analog converter, which is supposed to reside in the output stage of the chip, is not included here. Instead, it must be attached outside the chip when implementing an audio system.

VI. CONCLUSIONS

In this paper, the architecture of a low-frequency audio processor tuned for MP3 decoding has been presented. To achieve low operating frequency, we exploited fixed-point arithmetic and reduced the cycle count using a new instruction set optimized for the algorithm. To minimize the bit-width of fixed-point arithmetic while preserving precision, dynamic scaling is proposed and implemented. In addition, the SMAC instruction that can perform 32 MAC operations serially in a pipelined manner and the partially symmetric addressing mode with corresponding address generation unit structure are proposed to reduce the cycle count. The resulting cycle count required to decode an MP3 frame is so small that the proposed processor can operate at 12.8 MHz, which is the lowest among programmable implementations, while decoding in real-time.

ACKNOWLEDGEMENTS

This work was supported (in part) by the Korea Science and Engineering Foundation through the MICROS center at KAIST, Korea.

REFERENCES

[1] G. Maturi, "Single chip MPEG audio decoder," *IEEE Transactions on Consumer Electronics*, vol. 38, no. 3, pp. 348-356, August 1992.

[2] F. –O. Witte, R. Backes, M. Klumpp, E. Schidlack and M. Ulrich, "Multipurpose audio signal compander," *IEEE Transactions on Consumer Electronics*, vol. 39, no. 3, pp. 260-268, August 1993.

[3] T. –H. Tsai, T. –H. Chen and L. –G. Chen, "An MPEG audio decoder chip," *IEEE Transactions on Consumer Electronics*, vol. 41, no. 1, pp. 89-96, February 1995.

[4] MAS3057D MPEG 1/2 Layer 2/3 Audio decoder, Micronas GmBH.

[5] S. Hong, B. Park, Y. Song, H. Seo, J. Kim, H. Lee, D. Kim and M. Song, "A full accuracy MPEG1 audio layer 3 (MP3) decoder with internal data converters," *Proceedings of the IEEE 2000 Custom Integrated Circuits Conference*, pp. 563-566.

[6] ISO/IEC 11172-3, "Information Technology – Coding of Moving Pictures and Associated Audio for Digital Storage Media up to 1.5 Mbits/s," Part 4: Compliance Testing.

[7] I. Koren, "Computer arithmetic algorithms," Prentice-Hall International, Inc., 1993

BIOGRAPHY

Yongseok Yi received the B. Tech. and M.S. degree in electrical engineering from Korea Advanced Institute of Science and Technology (KAIST), Taejon, Korea in 1999 and 2001, respectively. Currently, he is a Ph.D. student in the Department of Electrical Engineering and Computer Science at KAIST. His research interests include VLSI design for signal processing and communication. He is a student member of the IEEE.

In-Cheol Park received the B.S. degree in Electrical Engineering from Seoul National University in 1986, the M.S. and Ph.D. degrees in Electrical Engineering from KAIST in 1988 and 1992, respectively. From May 1995 to May 1996, he worked at IBM T.J. Watson Research Center, Yorktown, New York as a postdoctoral member of the technical staff in the area of circuit design. He joined KAIST in June 1996 as a Professor in the Department of Electrical Engineering and Computer Science. He is a corecipient of the Best Paper Award for 1999 from IEEE International Conference on Computer Design. His current research interest includes CAD algorithms for high-level synthesis and VLSI architectures for general-purpose microprocessors. He is a member of the IEEE.