A DSP-Enhanced 32-bit Embedded Microprocessor

Hyun-Gyu Kim[1,2] and Hyeong-Cheol Oh[3]

[1] Dept. of Elec. and Info. Eng., Graduate School, Korea Univ., Seoul 136-701, Korea
[2] R&D center, Advanced Digital Chips, Seoul 135-508, Korea
babyworm@gmail.com
[3] Dept. of Info. Eng., Korea Univ. at Seo-Chang. Chung-Nam 339-700, Korea
ohyeong@korea.ac.kr

Abstract. EISC (Extendable Instruction Set Computer) is a compressed code architecture developed for embedded applications. In this paper, we propose a DSP-enhanced embedded microprocessor based on the 32-bit EISC architecture. We present how we could exploit the special features, and how we could overcome the deficits, of the EISC architecture to accelerate DSP applications with a relatively low hardware overhead. Our simulations and experiments show that the proposed DSP-enhanced processor reduces the average execution time of the DSP kernels considered in this work by 47.8% and the DSP applications by 29.3%. The proposed DSP enhancements cost about 10300 gates and do not increase the clock frequency. The proposed DSP-enhanced processor has been embedded in an SoC for video processing and proven in silicon.

Keyword: DSP-enhanced microprocessor, SIMD, hardware address generator, register extension, embedded microprocessor

1 Introduction

As more and more DSP (digital signal processing) applications run on embedded systems in recent years, it has become one of the most important tasks for embedded microprocessors to accelerate DSP applications. This trend is being reflected on the most successful embedded processors in the market: ARM cores added saturation arithmetic in ARMv5TE and SIMD (Single Instruction Multiple Data) instruction set in ARMv6 [1]; and MIPS cores adopted an application-specific-extension instruction set for DSP and 3D applications [2]. The capabilities of accelerating DSP accelerations should be implemented with as little hardware overhead as possible, since most embedded microprocessors target on the cost and power sensitive markets.

In this paper, we introduce a DSP-enhanced embedded microprocessor based on the EISC (Extendable Instruction Set Computer) architecture [3–5]. EISC is a compressed code architecture developed for embedded applications. While achieving high code density and a low memory access rate, the EISC architecture uses a novel and terse scheme to resolve the problem of insufficient immediate operand fields of the compressed code RISC machines. We present how we could exploit the special features of the EISC architecture to accelerate DSP applications with a relatively low hardware overhead. Since EISC also has deficits in processing DSP applications like any other compressed code architectures, we propose various schemes to overcome these deficits of the EISC.

In order to seek proper enhancements, we analyze the workload of benchmark programs from Mediabench [6] on the 32-bit EISC processor, called *base* below. The processor *base* is a 5-stage pipelined implementation of the non-DSP 32-bit instruction set of EISC [5]. Based on the profiling results, we modify the *base* by carefully choosing and adding DSP supporting instructions, such as SIMD MAC and saturation arithmetic. We also adopt and add various schemes for signal processing, such as a support for correcting radix point during the fixed-point multiplications, a hardware address generator for supporting memory accesses efficiently, and a scheme for increasing the effective number of general-purpose registers (GPRs).

Our simulations and experiments show that the proposed DSP-enhanced processor reduces the average execution time of the DSP kernels considered in this work by 47.8% and the DSP applications by 29.3%. The proposed DSP enhancements cost approximately 10300 gates only and do not increase the clock frequency.

This paper is organized as follows: In Section 2, an overview of the EISC architecture is presented. Section 3 describes the schemes we propose for enhancing DSP capabilities. Section 4 presents our evaluation results and an SoC that has been developed based on the proposed processor. Section 5 concludes the paper.

2 The EISC Architecture

In an embedded microprocessor system, code density and chip area are two major design issues, since these costs are more important in this area. However, many 32-bit embedded microprocessors suffer from poor code density. In order to address this problem, some RISC-based 32-bit microprocessors adopt 16-bit compressed instruction set architectures, such as ARM THUMB [7] and MIPS16 [8]. This approach provides better code density but needs some mechanisms to extend the insufficient immediate field and to provide backward compatibility with their previous architectures, which can result in extra hardware overhead. Moreover, these architectures have difficulty in utilizing their registers efficiently in the compressed mode [7, 8].

The EISC architecture takes a different approach for achieving high code density [3, 4]. EISC uses efficient fixed length 16-bit instruction set for 32-bit data processing. To resolve the problem of insufficient immediate operand fields in a terse way, EISC uses an independent instruction called `leri`, which consists of a 2-bit opcode and a 14-bit immediate value. The `leri` instruction loads immediate value to the special register called ER (extension register), and the value in the ER is used for extending the immediate field of a related instruction to make a long immediate operand.

By using the `leri` instruction, the EISC architecture can make the program code more compact than the competing architectures, ARM-THUMB and MIPS16, since the frequency of the `leri` instruction is much less than 20% in most programs. In [3], the code density of the EISC architecture was evaluated to be 6.5% higher than that of ARM THUMB and 11.5% higher than that of MIPS16 for various programs considered in [3]. In our experiments, the static code sizes of *base* are 18.9% smaller than those of ARM9-TDMI for the programs in Mediabench. As the `leri` instruction is just used to extend the immediate field of a related instruction, it can be a performance burden for the EISC processor. To overcome this deficit, EISC uses *leri instruction folding method* explained in [10].

As EISC uses fixed length 16-bit instruction set, the instruction decoder for the EISC architecture is much simpler than that of the CISC(Complex Instruction Set Computer) architecture. In addition, the EISC does not suffer from the overhead for switching its processor mode, which is often needed to handle long immediate values or complicated instructions in the compressed code RISC architectures. Moreover, the EISC architecture reduces its data memory access rate by fully utilizing its 16 registers while the competing architectures can access limited number of registers in the compressed mode. In [4], the data memory access rate of the EISC is 35.1% less than that of the ARM THUMB and 37.6% less than that of MIPS16 for the programs considered in [4]. Thus, the EISC can reduce both instruction references and data references. Reducing the memory accesses would bring reduction in power consumption related to the memory accesses and also lessen the performance degradation caused by the speed gap between the processor and the memory.

3 DSP Enhancements for the EISC Processor

In this paper, we propose several DSP enhancements for developing a DSP-enhanced EISC processor with as little extra hardware cost as possible. We develop instructions for enhancing DSP performances: instructions for SIMD operations with the capability of saturation arithmetic; and instructions for generating addresses and packing, loading, and storing media data. We try to minimize the overheads for feeding data into the SIMD unit [11]. We develop the enhancements so that they can be realized within the limited code space, since the processor uses 16-bit instructions (even though it has a 32-bit datapath.)

3.1 DSP Instruction Set

Since the data in the multimedia applications are frequently smaller than a word, we can boost up the performance of the processor *base* by adopting a SIMD architecture. When the SIMD operations are performed, however, some expensive packing operations are also performed for arranging the data to the

SIMD operation unit [11]. In order to reduce the number of packing operations, the processor should support various types of packed data. However, since the code space allowed for the DSP-enhancements is limited as mentioned above, we focus on accelerating the MAC operations which are apparently the most popular operations in the DSP applications. We implement two types of MAC operations shown in Fig. 1: the parallel SIMD type and the sum-of-products type. The parallel SIMD type of MAC operation, shown in Fig. 1(a), is much more efficient than any other types of operations for processing the arrays of structured data such as RGBA formatted pixel data. On the other hand, the sum-of-products type of MAC operation, shown in Fig. 1(b), is efficient for processing general data arrays.

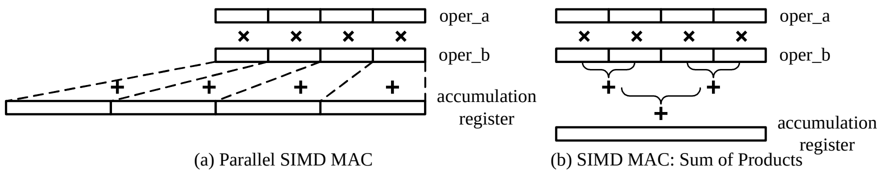

Fig. 1. Two types of the SIMD MAC operations supported in the proposed DSP-enhanced processor.

We also adopt the instructions for the saturation arithmetic which is often used in DSP applications. Unlike the wrap-around arithmetic, the saturation arithmetic uses the maximum or the minimum value when it detects an overflow or an underflow.

In the signal processing, the fixed-point arithmetic is commonly used because it is much cheaper and faster than the floating-point arithmetic. During the multiplication of two 32-bit fixed-point numbers, the results are stored in the 64-bit multiplication result register (MR). Since the position of radix point in a fixed-point number can be changed during multiplications, we need to select a part of the 64-bit multiplication result to make a 32-bit fixed-point number. For that purpose, *base* would require a sequence of five instructions looking like the one in the first box shown below. We propose to use an instruction with the mnemonic `mrs` (*multiplication result selection*), as shown below.

```
mfmh   %Rx           # move MR(H) to GPR(Rx)
mfml   %Ry           # move MR(L) to GPR(Ry)
asl    DP, %Rx       # DP-bit shift left Rx
lsr    (32-DP), %Ry  # (32-DP)-bit shift right Ry
or     %Rx, %Ry      # Ry = Rx | Ry
```

↓

```
mrs      DP, %Ry
```

3.2 Accelerating Address Generation

DSP applications usually perform a set of computations repeatedly for streaming data that have almost uniform data patterns. In this section, we introduce a loop-efficient hardware address generator that can accelerate the memory-addressing processes. We intend to use this address generator for accelerating DSP applications, including the memory-copy operations, with low cost.

The proposed address generator has a capability of handling the auto-increment addressing mode which is usually used for various applications including the memory-copy operations. As DSP operations use a loop for computing processes to handle streaming data, the proposed address generation unit is equipped with a comparator to detect the end-offset for loop. The address generator is also intended for supporting other special memory-addressing modes, such as the wrap-around incremental addressing mode for a virtual circular buffer and the bit-reversal addressing mode for transform kernels. We only support the post-modifying scheme because it is more popularly used [12].

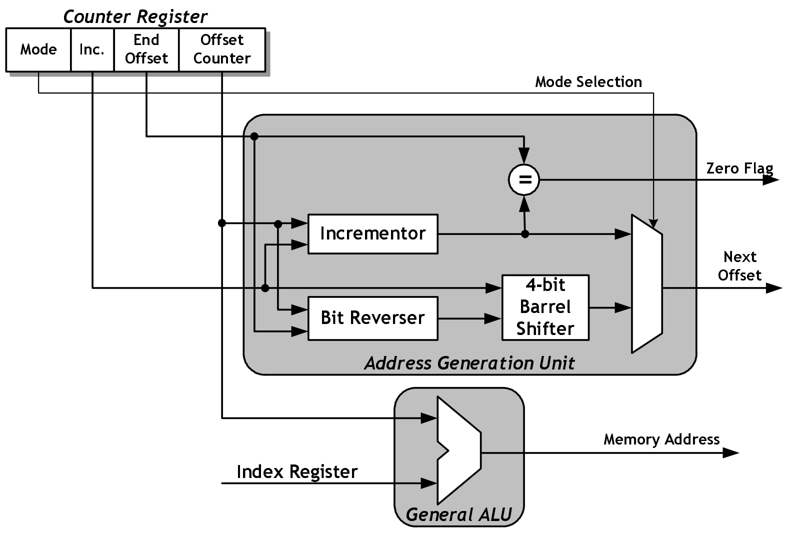

Fig. 2. Proposed address generation unit.

Since the hardware complexity is important, the proposed address generator only produces the small sized offset instead of the entire addresses. Fig. 2 shows the block diagram of the proposed address generator and the counter register. The control register is used to control the address generator and holds a control word regarding generation pattern, offset counter, end offset for looping, and incremental offset. Since we use a post-modifying scheme, the ALU uses the previously generated offset counter to make a memory address, while the incrementor uses it to generate the next offset. The generated offset is written back to the offset counter field in the counter register.

3.3 Register Extension

Almost DSP routines use many coefficients and temporary values simultaneously. However, most embedded microprocessors do not have enough registers to hold those values. A general approach for resolving this problem of shortage of registers is to spill the contents of registers into the memory. However, media data often take the form of data streams, in which case they are temporarily stored in the memory, continuously read into the processor, and used only a few times. Therefore, media applications running on an embedded processor often suffer from significant performance losses due to the frequent register spilling processes.

We propose a register extension scheme that increases the effective number of registers by adopting the idea of shadow register to hold temporary values in the registers. The idea of shadow register has been used in various aspects, such as the context switching latency [13]. We find that this idea is also very useful for embedded processors which have limited code spaces, such as the EISC processors, since it is hard for them to allocate additional bits for register indexing [9].

For the register extension scheme, we divide the register file into two parts: one part consists of a set of smaller register files, *register pages*, and another part consists of eight registers to be shared among register pages. Each register page consists of eight registers. We use three bits in the status register to select active register page. Thus we can use up to seventy two registers, eight shared registers and eight register pages with eight registers, in the proposed processor when whole register pages are implemented. The use of register pages decreases memory traffic due to memory spilling.

4 Evaluations

The designed DSP-enhanced microprocessor has been modeled in Verilog HDL as a synthesizable RTL model. We synthesize the model using the Samsung STD130 0.18μm CMOS standard cell library [14] and the Synopsys DesignCompiler. The critical path delay is estimated by using the Synopsys PrimeTime under the worst-case operating condition (1.65V supply voltage and $125\,^{\circ}$C junction temperature.) The results are summarized in Table 1. As shown in Table 1, the critical path delay is almost the same even through some units are added to accelerate DSP applications. The enhancements we add cost about 10270 equivalent gates, most of which are used for the shadow register file and the SIMD MAC unit. The others cost less than 1000 gates per each unit.

In order to evaluate the performance of the proposed architecture, we use DSP kernels used commonly in many DSP applications: IMDCT (inverse modified discrete cosine transform), FFT (fast Fourier transform), DCT (discrete

Table 1. Cost and power performance of the enhancements

	Area [equi. gates]	Critial Path Delay [ns]
Base processor	56852	6.25
Proposed processor	67122	6.19

cosine transform), and FIR (finite impulse response) filter. The IMDCT routine is used for reducing aliases in high fidelity audio coding applications including MP3, and AAC. The DCT is commonly used in the image compression. The FFT and FIR filter are the most common DSP functions in many DSP applications. We use the verilog-HDL model with the perfect memory model during the performance evaluation.

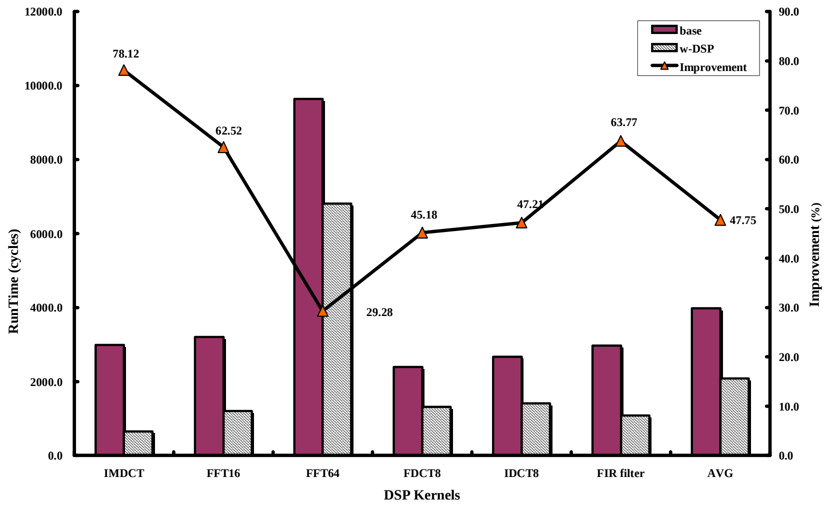

Fig. 3. Performance gain for popular DSP kernels.

The results of the experiments are summarized in Fig. 3. The proposed DSP architecture reduces the average execution time of the DSP kernels considered by 47.8%. For IMDCT, the use of the `mac` and `mrs` results in the reductions of the computation time. Furthermore, the use of the register extension scheme reduces the number of memory accesses by 9%. For FFT kernels, the performance improvement is mainly due to the parallel SIMD MAC operations. However, we observe a limited performance enhancement for the 64-point FFT, since it takes too much time to process the data packing and the memory accesses for spilling temporary values. For DCT and FIR filter, the instruction for sum of products is used efficiently and results in the performance gain. Moreover, the register extension scheme reduces memory operations in the FIR filter application. In our experiments, the coefficients for an 8-TAP FIR filter are loaded just once during the processing.

We also experiment with real DSP applications such as MP3 decoding program which uses MAD (MPEG audio decoding) library, ADPCM (Adaptive Differential Pulse Code Modulation) encoding program, and JPEG image decoding program. While selecting the functions to be optimized, we analyze the DSP applications to identify functions which take a large portion of the run time. In the experiments, a perfect (zero-wait) external memory is also assumed. The input data that we use and the selected functions are summarized in Table 2.

Table 2. Input data and selected kernels for the considered DSP applications.

DSP application	**Input data**	**Selected kernel(s)**
MP3 decoding	8.124-second stereo MP3 sample, 44.1KHz sampling rate, 192bps bit rate	*imdct_l* *subband synthesis*
JPEG decoding	64 × 64 × 24b jpeg file	*Huffman decode*
ADPCM encoding	3.204-second mono PCM sample, 8KHz sampling rate, 16-bit little endian	*ADPCM encode*

We measure the clock counts. The results are summarized in Table 3. For MP3 decoding application, optimized *imdct_l* and *subband synthesis* kernels reduce the run time by 30.8%. It means that the proposed DSP-enhanced processor is able to decode a high-fidelity MP3 audio at 31.1MHz, while the processor *base* has to be clocked at 45MHz to perform the same job. While optimizing the JPEG decoder, `cntl` instructions and SIMD saturate add instructions are used to optimize *Huffman decode* and the functions related on the variable length coding. In the case of the ADCPM encoding application, we reduce the execution time by 36.9% using the address generator and saturate add instructions. As a result, the proposed processor reduces the average execution time of the DSP applications considered by 29.3%.

Table 3. Performance gain for DSP applications.

DSP applications	***Base processor***	***Proposed processor***	**Improvement**
MP3 decoding	365,827,110	252,923,593	30.9%
JPEG decoding	4,507,509	3,597,074	20.2%
ADPCM encoding	1,914,368	1,208,188	36.9%

The proposed DSP-enhanced processor has been embedded in an SoC for video processing [15] and proven in silicon. Fig. 4 shows the layout of the SoC.

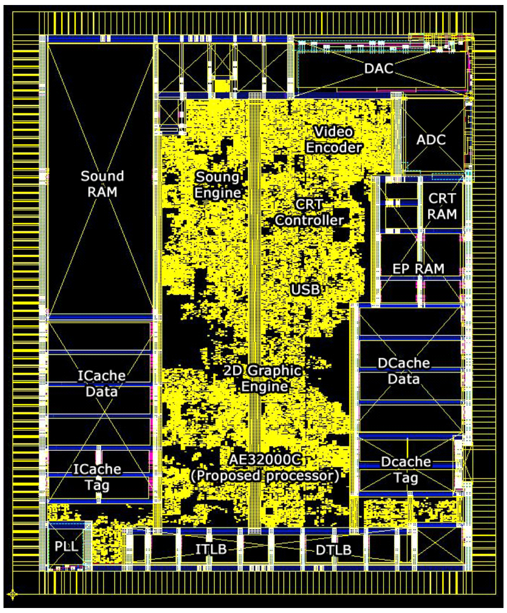

Fig. 4. Layout of the SoC equipped with the proposed DSP-enhanced EISC processor.

The SoC is equipped with a 2D graphics engine, a sound engine, a video encoder, a USB1.1 device controller, a four-channel DAC/ADC, and other peripherals including four DMAs, a memory controller, four UARTs, an I2S, a key-pad controller, an interrupt controller, a two-channel watchdog timer, a PWM, and a GPIO.

5 Conclusions

In this paper, we have introduced a DSP-enhanced embedded microprocessor based on the EISC architecture. In order to accelerate DSP application with as little extra hardware as possible, we propose various enhancement schemes: some schemes exploit the special features of the EISC, including the `leri` instruction; and some schemes are to overcome the inherent deficits of the EISC, including the insufficiency of the instruction bits and the insufficiency of GPRs.

We adopt the SIMD architecture and tailor it to reduce the hardware complexity and the packing overhead. To improve the performance of SIMD architecture, we propose a loop-efficient address generation unit. The proposed address generation unit is designed to support commonly used memory addressing modes in DSP applications with low hardware complexity. We also adopt a register extension scheme to reduce performance degradation due to the register spilling.

The proposed DSP-enhanced processor has been modeled in Verilog HDL and synthesized using a 0.18μm CMOS standard library. The proposed DSP enhancements cost about 10300 gates and not increase the clock frequency. Our simulations and experiments show that the proposed DSP-enhanced processor reduces the execution time of the DSP kernels considered in this work by 47.8% and the DSP applications by 29.3%. The proposed processor has been embedded in an SoC for video processing and proven in silicon.

Acknowledgements

The authors wish to acknowledge the CAD tool support of IDEC (IC Design Education Center), Korea and the financial support of Advanced Digital Chips Inc., Korea. The authors would also like to thank the anonymous reviewers for their valuable comments.

References

1. Francis, H.: ARM DSP-Enhanced Instructions White Paper, http://arm.com/pdfs/ARM-DSP.pdf
2. MIPS Tech. Inc.: Architecture Set Extension, http://www.mips.com/content/Documentation/MIPSDocumentation/ProcessorArchitecture/doclibrary
3. Cho, K.Y.: A Study on Extendable Instruction Set Computer 32 bit Microprocessor, J. Inst. of Electronics Engineers of Korea, **36-D(55)** (1999) 11–20
4. Lee, H., Beckett, P., Appelbe, B.: High-Performance Extendable Instruction Set Computing, Proc. of 6th ACSAC-2001 (2001) 89–94
5. Kim, H.-G., Jung, D.-Y., Jung, H.-S., Choi, Y.-M., Han, J.-S., Min, B.-G., Oh, H.-C.: AE32000B: A Fully Synthesizable 32-bit Embedded Microprocessor Core, ETRI Journal **25(5)** (2003) 337–344
6. Lee, C., Potkonjak, M., Mangione-Smith. H.: MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems, MICRO-30(1997) 330–335
7. ARM Ltd.: The Thumb Architecture Extension, http://www.arm.com/products/CPUs/archi-thumb.html
8. Kissell, K.D.: MIPS16: High-density MIPS for the Embedded Market, Technical Report, Silicon Graphics MIPS Group (1997).
9. Park, G.-C., Ahn, S.-S., Kim, H.-G., Oh, H.-C.: Supports for Processing Media Data in Embedded Processors, Poster Presentation, HiPC2004 (2004).
10. Cho,K.Y., Lim, J.Y., Lee, G.T., Oh, H.-C., Kim, H.-G., Min, B.G., Lee, H.: Extended Instruction Word Folding Apparatus, U.S. Patent No.6,631,459, (2003)
11. Talla, D., John, L.K., Buger, D.: Bottlenecks in Multimedia Processing with SIMD Style Extensions and Architectural Enhancements, IEEE Tras. of Comp. **52(8)** (2003) 1015–1011
12. Hennessy, J.L, Patterson, D.A.: Computer Architecture; A Quantitative Approach 3rd Ed., Morgan Kaufmann Publishers (2003)
13. Jayaraj, J., Rajendran, P.L., Thirumoolam, T.: Shadow Register File Architecture: A Mechanism to Reduce Context Switch Latency, HPCA-8 (2002) Poster Presentation
14. Samsung Electronics: STD130 0.18um 1.8V CMOS Standard Cell Library for Pure Logic Products Data Book, Samsung Electronics (2001)
15. Advanced Digital Chips Inc.: GMX1000: A High Performance Multimedia Processor User Manual, Advanced Digital Chips Inc. (2005)