Application Specific Processor Design for H.264 Decoder with a Configurable Embedded Processor

Jin Ho Han, Mi Young Lee, Younghwan Bae, and Hanjin Cho

An application specific processor for an H.264 decoder with a configurable embedded processor is designed in this research. The motion compensation, inverse integer transform, inverse quantization, and entropy decoding algorithm of H.264 decoder software are optimized. We improved the performance of the processor with instruction-level hardware optimization, which is tailored to configurable embedded processor architecture. The optimized instructions for video processing can be used in other video compression standards such as MPEG 1, 2, and 4. A significant performance improvement is achieved with high flexibility. Experimental results show that we could achieve 300% performance for the H.264 baseline profile level 2 decoder.

Keywords: Embedded processor, application-specific processor, configurable processor, H.264 video decoder, instruction selection algorithm.

Manuscript received Jan. 12, 2005; revised June 27, 2005.

The material in this work was presented in part at IT-SoC 2004, Seoul, Korea, Oct. 2004.

Jin Ho Han (phone: +82 42 860 6558, email: soc@etri.re.kr), Mi Young Lee (email: sharav@etri.re.kr), Younghwan Bae (email: yhbae@etri.re.kr), and Hanjin Cho (email: hjcho@etri.re.kr) are with Basic Research Laboratory, ETRI, Daejeon, Korea.

I. Introduction

The demand of multimedia communications on mobile and portable applications is growing nowadays. To realize multimedia communications, implementing a video compression standard is essential in any multimedia processing system-on-a-chip (SoC). There have been reports on the very large-scale integration (VLSI) implementation of MPEG-4 video recently. The emerging efficient H.264 or MPEG-4 Part 10 standard can greatly reduce the bandwidth and storage requirements for multimedia data. The VLSI implementation of H.264 is a challenge since an H.264 baseline decoder is approximately three times more complex than an H.263 baseline decoder [1].

With experience from the design of previous video compression standards such as MPEG 1, 2, and 4, we can re-use the designed IPs to reduce the design time. Implementational flexibility is an important factor of concern for SoC designs. Since the traditional hardwired design is less flexible, the processor-based implementation is a preferred choice. VLSI implementation can be categorized into three types, hardwired, digital-signal-processor-based, and hybrid. To achieve higher performance with flexibility, the hybrid architecture has been proposed, where operation-intensive functions are implemented with hardwired blocks, while other functions of less complexity are implemented with software on an application specific instruction processor [2], [3]. H.264 on OMAP Solution, which combined an ARM processor with TI's digital signal processor, implemented every function with software using an accelerated instruction set for multimedia

processing while keeping the flexible software structure [4]. However, it is worthwhile to point out that OMAP is not designed only to the embedded processor for video compression. There are many instructions unused in H.264 video processing. The cost is too high to be attractive so the advent of newly standardized video decoding/encoding such as H.264 has created the need for an application specific embedded processor to accelerate the specific function. In this paper, we describe the implementation of an H.264 video decoder using a configurable embedded processor.

This paper is organized as follows. In section II, we summarize the new processor design methodology using a configurable embedded processor. In section III, we give an overview of the H.264 decoder software. In section IV, we describe the instruction set and processor architecture specific to H.264 decoder application. Finally, we summarize the implementation results.

II. Design Methodology

The proposed processor is implemented using a configurable embedded processor. A new configurable embedded processor called Xtensa has been recently developed by Tensilica.

Figure 1 shows the architecture of the Xtensa processor. The Xtensa processor consists of a base instruction set architecture (ISA) feature that includes the basic instruction set, an extensible function that is able to add a user-defined instruction set, and configurable and optional functions that are configurable to the processor architectures, such as the interface options, memory subsystem options, and OS supports. The Tensilica Instruction Extension (TIE) language is used to describe a new instruction set and new registers, as well as execution units which can then be automatically added to the Xtensa processor.

The processor is a traditional reduced-instruction-set-computer-type processor with five-stage pipelines. Figure 2 shows the five pipeline stages: instruction fetch, instruction decoding, execution, memory access, and write back. When a new instruction is added, new decode, pipe control and coprocessor registers, and coprocessor ALU blocks are added in the processor data path, and address generation, exception resolution, and write back block are modified.

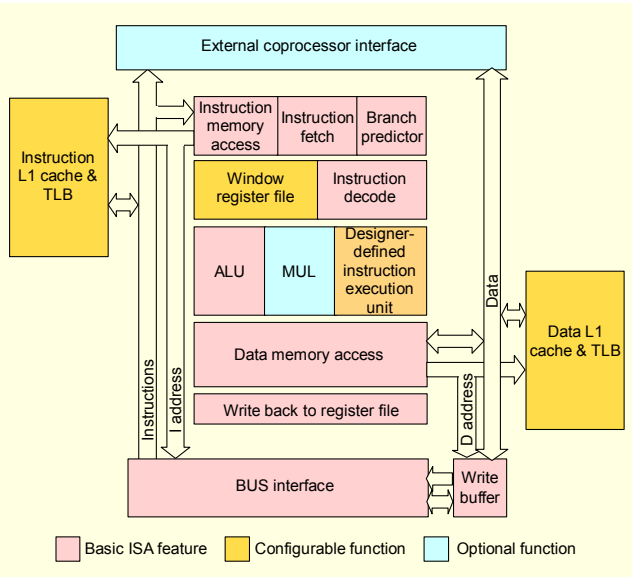

Fig. 1. Xtensa processor.

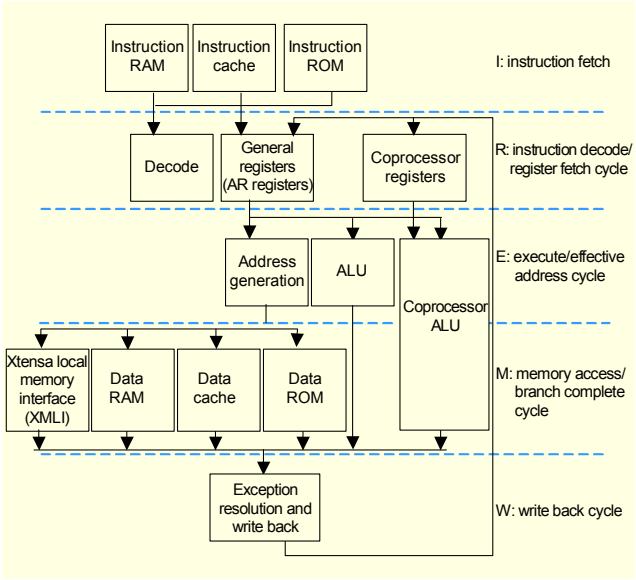

Fig. 2. The pipeline structure.

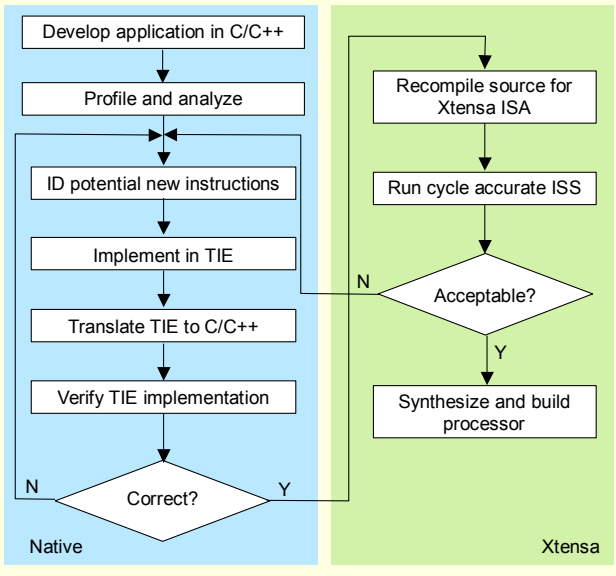

Fig. 3. The design methodology.

Based on the processor architecture described above, we can generate the instruction set simulator, compiler, and register transfer level (RTL) code of the processor which includes the new instruction set. Furthermore, using the simulator, compiler, and RTL code, we can analyze the run cycles after the application software runs on the processor. Also, we can know the die area and the power consumption of the processor.

Because of this functionality, we can use the new design methodology as shown in Fig. 3. Using the simulator and compiler, of which the processor is composed of the basic instruction set, the application software programmed in C/C++ is compiled and analyzed. And a new instruction that can reduce the run cycles of the application software is described in TIE. After generating the compiler and simulator, of which the processor is composed of extended instructions, the application software is compiled and analyzed again. This flow can repeat until the processor performance is satisfactory for the application software.

The design methodology can trade off between the performance and cost of the proposed instruction set. And we can estimate the exact number of run cycles of the application software.

III. Overview of H.264 Decoder

The H.264/AVC video coding standard has been introduced with significant enhancements in both video coding efficiency and flexibility over a variety of network domains. In a video coding layer (VCL), some of the important enhancements are the use of a small block-size (4 × 4) exact-match transform, adaptive in-loop de-blocking filter, and motion-prediction capability.

As shown in Fig. 4 after receiving the data from network abstraction layer (NAL), entropy decoded, inverse quantized and inverse transformed data are created and added to the predicted data from the previous frames depending upon the header information. Then, the original block can be obtained after the de-blocking filter [2].

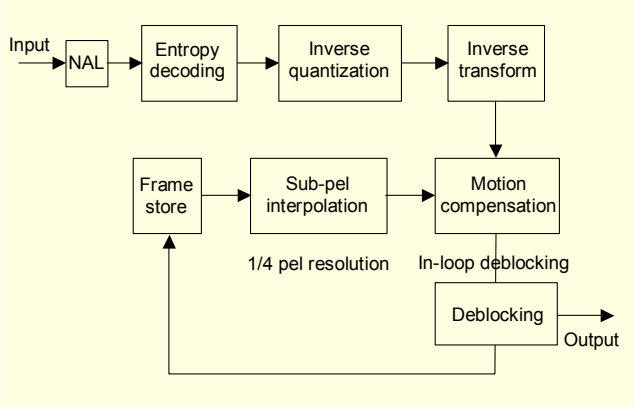

Fig. 4. H.264 video decoder.

H.264 defines three types of profiles. The baseline profile is the simplest and supports intra- and inter-coding, as well as entropy coding with context-adaptive variable-length coding (CAVLC). The distribution of time complexity among major subsystems was analyzed, and loop filtering, interpolation, inverse transform, bit stream parsing, and entropy decoding are the order of the averaged time consuming parts.

IV. Processor Architecture

1. Numbering and Ordering of References

The processor architecture consists of many options such as OS support, memory map, cache policy, interrupts, debug interface, and except-instruction set. We mention the memory system, the instruction set architecture, which is dependent on a processor performance.

We proposed an embedded processor architecture, based on MIPS basic instruction set, which is specific to MPEG4 codec application software using a SimpleScalar simulation environment [5]. The memory system for H.264 decoder software has the same parameter as in [6] and [7] using the proposed instruction set. Table 1 shows the cache size and policy, memory interface width, and performance.

The research on the instruction set for an application-specific processor is focused on reducing the run time of the MPEG application algorithm, such as motion compensation/estimation, discrete cosine transform, variable-length decoding, and color space conversion [6], [8]. We propose the extended instruction set for the H.264 video compression standard which was announced in 2004.

First, we developed an H.264 video decoder for a baseline profile at level 2 in C language. The video format is CIF 15 frames/sec. We analyzed the execution times of the functions for the H.264 video decoder.

Based on the profiling result with general example data, we noticed that motion compensation, entropy decoding, and

Table 1. The proposed memory system.

Architecture feature	Parameter	Performance
Basic feature	5 stage	Hit rate: 0.90 IPC: 0.79
Cache	Cache size : 16kB Block size : 64 B Set associative : 2-set	
Memory interface width	128 bit	

inverse quantization and integer transform consume most of the time [6].

2. Motion Compensation

Motion compensation reconstructs the current macro block using a motion vector, which is the difference between the current macro block and the macro block of the previous frame. The difference value is quantized inversely and performed using an inverse integer transform. The major arithmetic included are the multiply and accumulation operations. Motion compensation performs interpolation operations for chrominance components. Table 2 shows the profiling results of a motion compensation function based on simulation cycles.

The main functions of motion compensation are get_block, mc_main, _mulsi3, and _divsi3. The get_block function calculates the block value by interpolating two 8-bit integer values. The mc_main function calculates the current block value by adding the reference block value to the value multiplied by the coefficient. The _mulsi3 function is a 32-bit signed multiplication function. The _divsi3 function is a 32-bit signed division function. They are inserted by the compiler because the processor does not have mulsi3, divsi3 instructions. Table 3 shows the proposed instructions for motion compensation.

Table 2. Motion compensaton.

Function	Simulation cycles/10 frames (Mcycles)
mc	1.03
mc_main	20.90
get_block	51.93
_mulsi3	26.31
_divsi3	9.49
Total simulation cycles	110.66

Table 3. The proposed instructions for motion compensation.

Instruction set	Description
Mul32	arr[31:0] = ars[15:0] × art[15:0]
CLP8	arr[31:0] = (ars < 0)?0 : { (ars > 0xff)?0xff : ars }
MULADD_COEFF	arr[63:32] = arr[63:32] + art[15:0] × C_0, arr[31:0] = arr[31:0] + art[47:32] × C_1

3. Entropy Decoding

H.264 supports two different methods for the final entropy encoding step: CAVLC is the standard method using simple variable length Huffman-like codes and codebooks. Table 4 shows the profile result of the entropy decoding function based on simulation cycles.

Table 4. Entropy decoding.

Function	Simulation cycles/10frames (Mcycles)
showbits	26.77
init_mb	21.53
code_from_bitstream_2d	5.88
_umodesi3	4.84
_udivsi3	4.12
readCoeff4×4_cavlc	3.93
misc	4.25
Total simulation cycles	71.32

Table 5. The proposed instructions for entropy decoding.

Instruction set	Description
showbits	bit position calculation in frame
MUL32	arr[31:0] = ars[15:0] × art[15:0]
code_from_bitstream	coefficient calculation
read_coeff_4×4	coefficient calculation

The showbits function calculates the bit position in any frame. The function has not many run cycles but many calling numbers. The instructions written for the showbits function are proposed. We proposed the instructions for a mod operation and a 32-bit unsigned division operation to reduce the run cycles of _umodesi3 and _udivsi3. Also we proposed instructions for code_from_bitstream and read_coeff_4×4 which have a coefficient calculation as shown in Table 5.

4. Inverse Quantization and Integer Transform

The basic algorithm of H.264 uses a separable transformation. The mode of operation is similar to that of JPEG and MPEG, but the transformation used is not an 8 × 8 discrete cosine transform (DCT), but a 4 × 4 integer transformation derived from the DCT. It can be computed using only additions, subtractions, and binary shifts. The actual video data is subtracted from the prediction. The resulting residual is transformed by inverse quantization. The coefficients of this transform are divided by a constant integer number. Table 6 shows the profiling results of the inverse quantization and integer transform (ITIQ) function based on simulation cycles.

Itrans and itiq_main are used in inverse integer transformation. The itrans instruction for itrans function, ih_luma instruction for ih_lumadc, and ih_chromadc instruction for ih_chromadc function are adaptive instructions. The iquant function has many 32-bit multiplications. Table 7

Table 6. Inverse quantization and integer transform.

Function	Simulation cycles/30 frames (Mcycles)
itiq	9.04
itiq_main	23.41
itrans	38.39
iquant	33.34
reset_coeff_buff_block	16.14
reset_coeff_buff	7.66
ih_chromadc	0.24
ih_lumadc	0.07
Total simulation cycles	128.29

Table 7. The proposed instructions for ITIQ.

Instruction set	Description
iTRANS	4×4 integer transform with special register
iH_LUMADC	Matrix arithmetic
iH_CHROMADC	Inverse hadamard arithmetic
MUL32	arr[31:0] = ars[15:0] × art[15:0]

shows the proposed instructions for ITIQ.

V. Implementation Result

We implemented processors with various instruction sets to trade off area and power. Table 8 summarizes the comparison results for the processors.

Processor I includes only basic instruction sets. Processor II includes the instructions proposed for motion compensation and entropy decoding, as well as the iTRANS instruction in addition to the instructions used in processor I. Processor III includes the MUL32 instruction in addition to the instructions used in processor II. In the case of processor III, although the gate counts are increased, it has the most reduced run cycles.

Table 8. Comparison of the processors.

Version	Unoptimized application code	Optimized application code		
		Processor I	Processor II	Processor III
Instruction set	Basic instruction set	Basic instruction set	Processor I +MC Instrs. +ENT Instrs. +iTRANS Instr.	Processor II +MUL32 Instr.
Run cycles (Mcycles)	304.7	224.7	150.04	115.3
Gate counts	25000	25000	40000	70000

We implemented processor III, finally. The processor runs motion compensation, ITIQ, and ENT of an H.264 baseline@L2 decoder of 10 frames/sec at a 115.3 MHz clock frequency. Table 9 shows the comparison results of the software run cycles between ARM9TDMI and the proposed processor.

The implemented processor is proved on an XT2000 board, an FPGA board dedicated to an Xtensa processor, as shown in Fig. 5. The application software is cross-compiled with program and data code at the host PC, and the compiled binary code is transferred to SDRAM through the debugging interface. We can monitor the results of the program through the serial interface on the host PC.

Fig. 5. FPGA board.

VI. Conclusion

We described an embedded processor design for an H.264 video decoder with a configurable embedded processor for wireless multimedia communication. We designed the processor architecture, which has a new instruction set and configured memory system. We designed the extended instruction set based on the profiling results. The extended instructions are specified for H.264 video decoder application software. The proposed processor is implemented with the configurable processor, Xtensa of Tensilica, Inc. Our improvement was carried out with a special design of instructions and the proper processor configuration. The overall improvement allows the software implementation of motion compensation, ITIQ, and ENT algorithms to be over 10 fps for a CIF sequence with low power, low cost, and great software flexibility.

Reference

[1] Michael Horowitz, Anthony Joch, Faouzi Kossentini, and Antti Hallapuro, "H.264/AVC Baseline Profile Decoder Complexity Analysis," *IEEE Trans Circuits and Systems for Video Technology*, vol. 13, no. 7, July 2003, pp. 704-716.

[2] H.264 JM Model, http://bs.hhi.de/~suehring/tml/

[3] Hyun-Gyu Kim, Dae-Young Jung, Hyun-Sup Jung, Young-Min Chio, Jung-Su Han, Byung-Gueon Min, and Hyeong-Cheol Oh, "AE32000B: A Fully Synthesizable 32-Bit Embedded Microprocessor Core," *ETRI J.*, vol.25, no.5, Oct. 2003, pp.337-344.

[4] J. Chaoui, K. Cyr, S. de Gregorio, J.-P. Giacalone, J. Webb, and Y. Masse, "Open Multimedia Application Platform: Enabling Multimedia Applications in Third Generation Wireless Terminals through a Combined RISC/DSP Architecture," *Proc. ICASSP*, vol. 2, 2001, pp. 1009-1012.

[5] Hoseok Lee, Jin Ho Han, Younghwan Bae, and Hanjin Cho, "Design of Embedded Processor Architecture Applicable to Mobile Multimedia," *J. the Institute of Electronics Engineers of Korea*, vol. 41-SD, no. 5, May 2004, pp. 71-80.

[6] Xiaosong Zhou, Eric Q. Li, and Yen-Kuang Chen, "Implementation of H.264 Decoder on General-Purpose Processors with Media Instructions," *Proc. SPIE Conf. on Image and Video Communications*, vol. 5022, 2003, pp. 224-235.

[7] Jung-Hoon Lee, Gi-Ho Park, and Shin-Dug Kim, "Dual Cache Architecture for Low Cost and High Performance," *ETRI J.*, vol.25, no.5, Oct. 2003, pp.275-287.

[8] Nathan Slingerland and Alan Jay Smith, *Multimedia Extensions for General Purpose Microprocessors: A Survey*, Report No. UCB/CSD-00-1124, Univ. of California at Berkeley, 2000.

Jin Ho Han was born in Korea on Feb 8, 1977. He received the BS and MS degrees in electrical engineering from Korea Advanced Institute of Science and Technology in 1998 and 2001. He joined Electronics and Telecommunication Research Institute (ETRI) in 2001, where he currently works in low power embedded processor design and SOC design methodology development as a project member.

Mi Young Lee was born in Korea on Oct. 3, 1976. She received the BS degree and in electronic engineering and the MS degree in information electronic engineering from Ewha Womans University in 1999 and 2001. She joined ETRI in 2001, where she currently works in low power multimedia SOC design and methodology development as a project member.

Younghwan Bae was born in Seoul, Korea on October 29, 1962. He received the BS and MS degrees in electronic engineering from Hanyang University in 1985 and 1987. He joined ETRI in 1987, where he works in developing CAD tools and design methodology for SOC.

Hanjin Cho was born in Seoul, Korea on July 8, 1960. He received the BS degree in electronic engineering from Hanyang University in 1982. He received the MS and PhD degrees in electrical engineering from New Jersey Institute of Technology in 1987, and from University of Florida in 1992. He joined ETRI in 1992, where he currently works in SOC design methodology development and wireless multimedia SOC design as a project manager.