Design of Application Specific Processor and Compiler for H.264 CAVLC Decoding

Jae-Jin Lee, Jun-Young Lee, MooKyoung Jeong, SeongMo Park and NakWoong Eum
Electronics and Telecommunications Research Institute,
Daejeon, 305-700, Korea
E-mail: {ceicarus, sollok, mookyoung, smpark, nweum}@etri.re.kr

Abstract **:** ASIPs are powerful solution which combines high performance of ASICs and flexibility of general purpose processors. This paper proposes a new application specific processor and compiler for CAVLC decoding and portable multimedia application. They are based on the 6-stage pipelined dual issue VLIW(Very Long Instruction Word) architecture, efficient CAVLC decoding instructions and compiler mapping techniques such as CKF(Compiler Known Function), Inline-Assembly and CGD(Code Generator Description). The proposed application specific processor whose gate-count is about 73K runs at 100MHz. Compared to the ARM966ES processor, the proposed method results in about 80% speed-up in terms of execution time and about 50% reduction in terms of hardware complexity without quality degeneration.

1. Introduction

ASIP(Application Specific Instruction Set Processor) is a powerful solution when the contradicting requirements such as performance and flexibility have to be jointly satisfied with a single task block.

H.264/AVC[1] video coding standard has gained more and more attention due to its improved coding efficiency. In contrast to the efforts of gaining higher coding efficiency, the complexity of H.264/AVC is more than previous standards[2]. Therefore, software-based real-time decoder requires more processors and faster algorithms. Especially, CAVLC decoding has been widely studied to decode the quantized transform coefficients efficiently. It uses several extensive dedicated code tables, and those tables are chosen by the context of previous blocks and symbols. In a H.264 baseline profile, the coefficients for a 4x4 block are mainly compressed by three syntax elements[1] such as coeff-token, total_zero, and run-before. Many CAVLC decoding methods have been presented so far. Implementation with the combinational logics, the VLSI architecture for reducing hardware costs and power consumptions, pattern-search method, and coeff-token decoding method with efficient memory access have been presented in [3][4][5], respectively. Typically, variable-length coding tables are used to decode coeff_token, total_zero, and run-before in a software implementation. This paper proposes new application specific architecture and compiler based on dual issue VLIW architecture and instruction-sets for efficient H.264 CAVLC decoding such as 'CLZ(Count Leading Zero)', 'DTZ(Decode Total_Zero)' and 'DRB(Decode Run-Before)'.

The rest of this paper is organized as follows: In chapter 2, application specific instruction-set processor and compiler are introduced. In chapter 3, the synthesis and simulation results of the proposed architecture and compiler are presented. Concluding remarks are presented in chapter 4.

This work was partly supported by the IT R&D program of MIC/IITA. [08MB2610, Multi-Format Multimedia SoC based on MPCore Platform]

2. Application Specific Instruction-Set Processor and Compiler

The proposed application specific architecture based on 6-stage pipeline such as PF(PreFetch), FE(FEtch), DC(DeCode), EX(Execution), MEM(MEMory), and WB(WriteBack) consists of dual issue VLIW core, separate program and data memory, register file consisting of 16 32-bit general purpose registers, and bus interface to access external memory. The pipeline is fully bypassed, i.e., instructions reading from register "R" can directly follow the instruction writing to the same register. The bypass logic ensures a consistent read access between the instructions and makes sure that the latest result for a register is read by the instruction. A bypass is required when an instruction "X" in the execute stage(EX) is producing a result that is read by the following instruction "Y". Instruction "Y" is at that time in the decode(DC) stage and requesting the result. A bypass allows instruction "Y" to access the result before it is actually written back into the main register file.

Fig. 1 provides a block diagram of the bypass logic. It can be seen that the main register file is accessed in the DC stage. The operands are immediately pushed into the pipeline registers "op1", "op2" and "op3". If a custom instruction would need these operands already in the DC stage, the values can also be written into a signal instead of a register. Thus, those signals can be used to any combinatorial data-path in the DC stage. In EX stage, the latest operand value is written into the signal "alu_in1", "shifter_in1" and "shifter_in2". Once the result is computed the "writeback_dst" operation need to be activated (not shown here) and the register address "BPR" as well as the writeback value "WBV" need to be

written. The MAC operation is very beneficial to speed-up many different type of applications. As shown in Fig. 2, the pipelined dual cycle MAC has been implemented by sharing basic multiplier unit. This results in efficient micro-architecture from an area cost point of view.

The Instruction-set of the proposed architecture consists of basic load/store, arithmetic, logic, branch and trap instructions, and custom instruction-sets which are specific to H.264 CAVLC decoding as shown in Fig. 3 and Table 1, respectively.

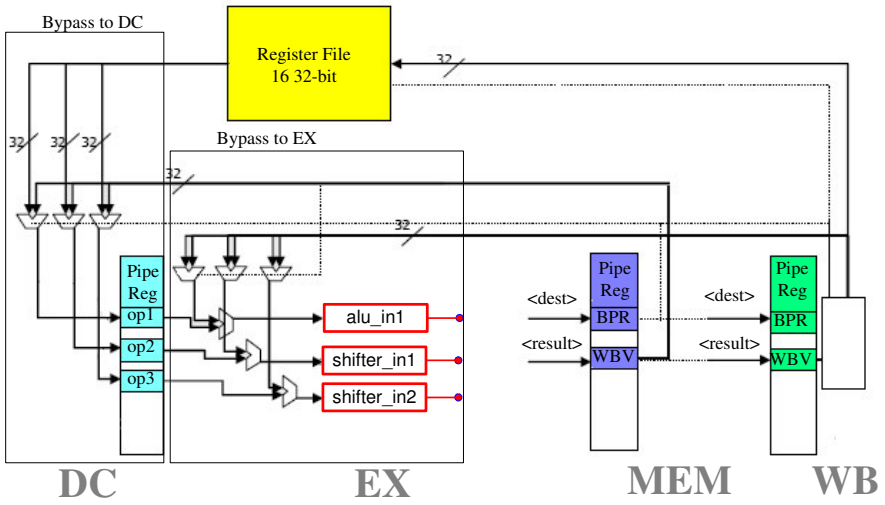

Fig. 1. Block diagram of bypass logic

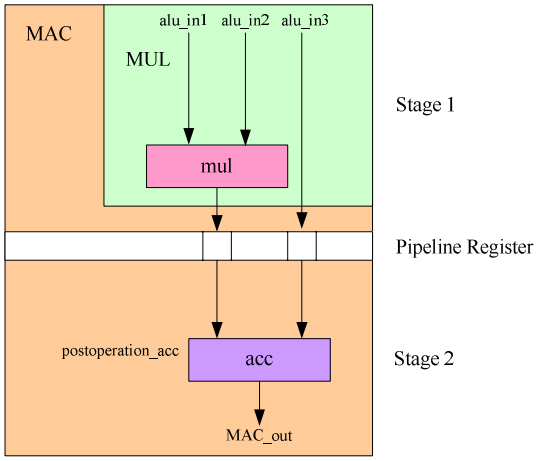

Fig. 2. Pipelined dual cycle MAC

Instruction-Set	Assembly Syntax
Instruction Registers	
Root	insn_set
insn_set	
custom_isa	instructions
trapi	"trap" ~" " ~imm
cmp_ri	"cmp" ~opcode ~" " src1 ~"=" src1 "," imm16
cmp_rr	"cmp" ~opcode ~" " dst ~"=" src1 "," src2
brai	"b" ~" " src1
bra	"b" ~cond ~" " src1 ~"," addr20
nop	"nop"
st_rr	opcode ~" " src2 "," src1 "," imm16
ld_rr	opcode ~" " dst "," src1 "," imm16
ldc_ri	"ldc" ~" " dst "," imm16
lui_ri	"lui" ~" " dst "," imm16
alu_rrri	opcode ~" " dst "," src1 "," src2 ~" " mode ~" " imm5
alu_rrrr	opcode ~" " dst "," src1 "," src2 ~" " mode ~" " src3
alu_rri	opcode ~" " dst "," src1 "," imm16
alu_rrr	opcode ~" " dst "," src1 "," src2
brau	"b" ~cond ~" " addr20

Fig. 3. Basic instruction-set

Table 1. Custom instruction-set

Instructions	Description
MUL	Singed multiplication
MIN/MAX	Min and max operation
ABS	Absolute value calculation
LOOP	Hardware-loop
CLZ	Count Leading Zero
DTZ	Decode Total-Zero
DRB	Decode Run-Before

In the H.264 baseline profile, quantized coefficients are encoded by five syntax elements such as 'Coeff_token', 'Sign of T1s', 'Level', 'Total_zero' and 'Run_before'. The fixed-length code tables for signs of T1s and level are regular and simple. However, the remaining three VLC(Variable-Length Code) tables for coeff_token, total_zero and run_before are irregular and very complex, and finally lead to performance bottleneck because of recursively repeated memory access in a software implementation. In order to reduce memory access, we designed special instructions such as "CLZ", "DTZ" and "DRB" shown in Table 1. Total_zero and run_before tables are efficiently implemented into application specific processor together with search algorithms. Fig. 4 shows 64-bit RUN_LEN and RUN_COD registers used to store run_before table[1] in the case of vlcmum > 6. Each element of length and code table is represented within 4-bits. In application C code, "DRB" instruction is accessed through Inline-Assembly code supported by C-compiler. Decoding of total_zero is performed in the same manner.

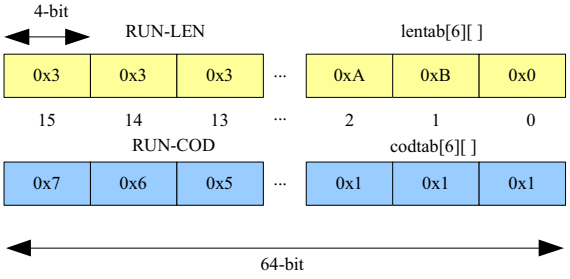

Fig. 4. 64-bit registers for run_before table in the case of vlcnum > 6

The coeff_token table is too big to be implemented into application specific processor. In the case of coeff_token decoding, we propose a new decoding method which drastically reduces memory access compared to the conventional method. The main idea is based on the fact that the pair of length and code value is unique and could be grouped by the number of leading zero. We first divide the table into groups according to the number of leading zero, and then rearrange the table by changing the table index from pair of TC and T1s to pair of length and code value. "CLZ" instruction is used to identify the group of encoded bit stream in a single cycle. We can decode any kind of coeff_token bit stream in maximum three times memory access using the proposed method without quality degeneration.

The memory controller of the proposed architecture is designed by considering synchronous memory access from a LISA model. A memory read or write access (transaction) for a synchronous memory access is separated into just two phases as shown in Fig. 5.

1. Request phase: A request for a read or write access is indicated.
2. Try phase: The data can be tried to be read from the memory. For a write access, it can be checked whether the data could be written or not.

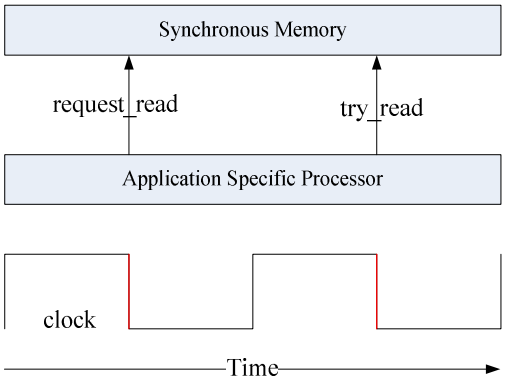

Fig. 5. Synchronous read access sequence

The compiler for the proposed architecture is developed using the C-compiler designer tool. "MIN", "MAX" and "ABS" instructions, "CLZ", "DRB" and "DTZ" instructions, and basic and dual cycle MAC instructions are mapped by CKF, Inline-Assembly and Matcher rules[6], respectively.

CKF stand for compiler known function. It is used to implement certain special instruction combination which would not usually be output by the compiler. The appropriate header file for the declaration of the function is automatically included by the compiler and usage of CKFs is similar to the normal function call. The advantage of CKFs over Inline-Assembly is that it gives more control to the C-compiler designer than to the user. Designers need not be aware of the assembly instructions that are required to implement the functionality.

The procedure of calling the assembly instructions within C code is known as Inline-Assembly. A block of Inline-Assembly code is similar to C function definition, except that the body of the function consists of a set of assembly instruction, and that invoking the function doesn't translate function call, but to an inlining of the assembly code.

A common technique to execute loops with zero-overhead is called hardware loop. Here, overhead means

instructions for loop counter increment, condition check, branch as well as pipeline stalls and delay slot. In the case of hardware loop, the exit loop condition, loop counter increment and branch are not performed in software through instructions but in hardware through dedicated logic. The design flow of a hardware loop is described as follows:

1. A loop initialize instruction first set the number of loop counter and the end address of the loop block.
2. If the fetch program counter has reached to the end of loop block and maximum number of iterations has not been reached, it is automatically set to the start address again for the next iteration

3. Synthesis and Simulation Results

The LISA(Language for Instruction-Set Architecture) processor design platform offers the possibility to generate structured RTL model by grouping operations into functional units. Each functional unit in the LISA model represents an entity or a module in the HDL model. The generated verilog RTL code for application specific processor is synthesized using Synopsys' Design Compiler based on TSMC 0.18μm cell library. Table 3 lists the synthesis results of application specific processor. Synthesis results show that the proposed architecture will reach 100MHz in 0.18μm technology.

We have verified functionality of the proposed architecture and compiler by replacing CAVLC decoding module of the existing VLSI implementation of H.264 video decoder platform[7] with the application specific processor proposed in this paper. Fig. 6 shows simulation results of the proposed architecture and ARM966ES processor whose approximate gate-count is 120K-140K at 200MHz for seven test video sequences. What can be seen is that we can achieve significant improvement in terms of processing cycle. The seven test video streams are 4:2:0 CIF (352x288) with QP=28 and 30frames.

Table 3. Synthesis results

Technology	TSMC 0.18μm	
Clock constraint	10 ns	
Operating conditions	typical	
Wireload model	default	
Area	VLIW core	53.0K
	Register file	19.8K
	Memory controller	0.4K
	Total	73.2K
Timing (ns)	9.92 (100 MHz)	

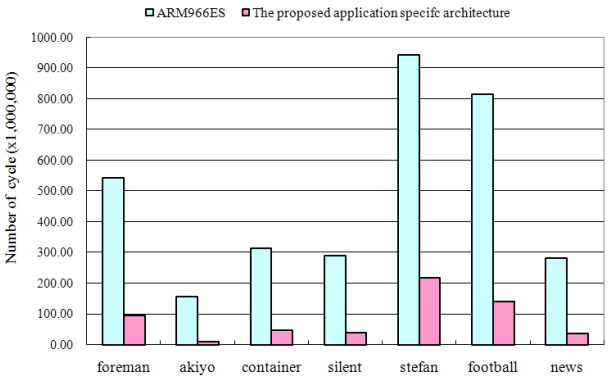

Fig. 6. Simulation results

4. Conclusions

This paper proposes a new application specific architecture/compiler. The proposed application specific processor whose gate-count is about 73K runs at 100MHz in 0.18μm technology. Compared to the conventional ARM966ES processor, the proposed method results in about 80% speed-up in terms of processing cycles and about 50% reduction in terms of hardware complexity without quality degeneration for the H.264 CIF test sequences. Future work will include more analysis on the power consumption of the proposed architecture.

References

[1] ISO/IEC 14496-10 International Standard (ITU-T Rec. H.264)

[2] M.Horowitz, A.Joch, F.Kossentini, and A.Hallapuro, "H.264/AVC baseline profile decoder complexity analysus," IEEE Trans.on Circuits and Systems for Video Tech., IEEE, vol.13, pp. 704-716, 2003

[3] W.Di, G.Wen, H.Mingzeng, and J.Zhenzhou, "A VLSI architecture design of CAVLC decoder," Int. Conf. ASIC, vol. 2, pp. 962-965, 2003

[4] H.-C.Chang, C.-C.Lin, and J.-I,Guo, "A novel low-cost high-performance VLSI architecture for MPEG-4 H.264/AVC CAVLC decoding," ISCAS, pp. 6110-6113, 2005

[5] Y.H.Moon, "A New Coeff-Token Decoding Method With Efficient Memory Access in H.264/AVC Video Coding Standard", IEEE Trans. Circuit and Systems for Video techl., vol 17, pp. 729-736, 2007

[6] Compiler Designer Reference Mannual, CoWare, 2007

[7] S.-M.Park et al. "VLSI implementation of H.264 Video Decoder for Mobile Multimedia Application," ETRI Journal, vol. 28, pp. 525-528, 2006