H.264/AVC CODEC: INSTRUCTION LEVEL COMPLEXITY ANALYSIS

Ce Xu, Thinh M. Le, Teng-Tiow Tay, MIEEE

Department of Electrical and Computer Engineering, National University of Singapore

ABSTRACT

H.264/AVC has been designed particularly to improve coding efficiency and network friendliness. In this paper, a simulator and profiler tool set based on the SimpleScalar framework [1] is developed to derive instruction level complexity of the H.264/AVC codec [2]. The reference software JM8.6 is analyzed. Arithmetic, logic, shift, control operations and memory bandwidth requirements for the full codec are presented. The results of various configurations on a tool-by-tool basis are compared and analyzed so that trade-off decisions can be made. Finally, a performance metric combining complexity, the objective quality and the bit-rate is presented to assist design space exploration.

1. INTRODUCTION

H.264/AVC [2] employs the same basic coding framework as that of MPEG2, H.263, and MPEG4. It achieves enhanced coding efficiency by adopting tools such as multiple reference frames, variable block size, *Content Adaptive Binary Arithmetic Coding* (CABAC), etc. These tools demand much higher computation capability. Efficient implementation targeting at real-time applications requires proper choice of the hardware platform. Traditional approaches are: general purpose processor, media enhanced *Digital Signal Processor* (DSP) and *application-specific integrated circuit* (ASIC) chip. It is therefore essential to identify the instruction level processing requirements to assist system design.

Priori works on complexity assessment have been presented in [3-5], and [7]. In [4], [5], complexity issue is addressed in terms of decoding/encoding time only. Horowitz *et al*. [3] estimated the complexity in machine cycles, but the analysis is limited to the baseline profile of the decoder. Their method on average underestimates the actual implementation complexity by a factor of 3-5. Saponara *et al.* [7] reported total access count and peak memory usage using the ATOMIUM tool set. However, the instruction level complexity in terms of arithmetic, logic, shift, and control operations were not addressed.

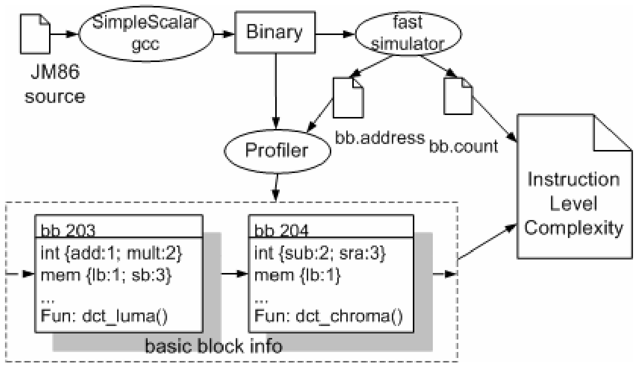

Figure 1: Simulation and profiling methodology.

In this work, a novel method is developed to report the encoder and decoder complexities in RISC-like instructions. As the reference software is non-optimized, all reported complexity will be higher than optimized implementations. The analysis focuses on functional unit requirements (integer and floating point ALU), memory access bandwidth and control logics as these are the main costs for both software and hardware realizations. H.264/AVC tools are examined on a tool-by-tool basis for the full codec. Computation complexity, objective quality (SNR-Y) and bit-rate saving are combined to form the complete metric for design space exploration.

The rest of the paper is organized as follows. Section 2 describes the simulation and profiling method. Section 3 shows the codec configurations. Section 4 and 5 provides the analysis of encoder and decoder complexity, respectively. Section 6 presents a performance metric incorporating complexity, quality, and bit-rate. Finally, conclusions are provided in Section 7.

2. SIMULATION/PROFILING METHODOLOGY

The H.264/AVC reference software JM8.6 is compiled to the SimpleScalar target, which is based on the popular MIPS IV ISA. We developed a tool set to gather the data and keep the simulation time practical. The strategy is to split the simulation into two stages: the fast simulator and the post-simulation profiler as depicted in Fig 1.

In the first stage, the user pre-configures a sampling period in terms of number of instructions (default 20 million for encoder, 1 million for decoder). Then fast functional simulation is performed and the simulator records each accessed basic block's address into a basic block index file (bb.address). The simulator also summarizes a basic block access count for every sampling period (bb.count). In the second stage, the profiler loads the program binary file and the basic block address file (bb.address) to generate detailed information for each basic block, including total number of instructions in the basic block, count of each instruction, and the function routine that the basic block belongs to by comparing the basic block's address with the text symbols in the binary file. The profiler then combines the basic block information with the basic block counts (bb.count) to generate various statistics for each sampling period, hence the entire simulation. This strategy introduces minimal simulation overhead and the simulation speed is measured 15 MIPS on a 2.8 GHz Pentium 4/Linux system.

3. CONFIGURATION

The reference software JM8.6 is designed to include many tools, to analyze the complexity on a tool-by-tool basis we performed multiple simulations. The detailed configurations for the codec are listed in Table 1. For instance, configurations 1 and 2 force intra coding, and configuration 2 further switches on all advanced tools such as CABAC. Each of the configurations 3-10 encodes the sequence in *IPPP...* order. All advanced tools are switched off and on for configurations 3 and 10, respectively. For configurations 4-9, each has one or two tools switched on. B-frame is not enabled in all the simulations. The Foreman testing sequence is in QCIF size and 50 frames are coded for each simulation.

4. COMPLEXITY: ENCODER

4.1. Overall Complexity

The SimpleScalar ISA is classified into six classes: load, store, unconditional branch, conditional branch, integer ALU, and floating point ALU. Figure 2 shows the percentage of instruction classes in each sampling period for the first 5 frames (configuration 6). Frame-to-frame boundaries can be directly observed from the distribution patterns. Fig. 3 shows the distribution on a per-frame basis over the entire 50 frames. It is worth noting the first frame, which is intra coded, requires the least number of operations whereas the complexity for frames 2, 3, 4 keeps increasing due to added motion estimation reference frame. All subsequent P-frames (4-7) show consistent complexity since maximum reference frame is 3.

Table 2 gives the RISC-like instruction complexity for configuration 10 (all tools switched on), including the first I-frame, subsequent P-frame and the average over the sequence. Fig. 3 gives the comparative average complexity of all the configurations. The encoder requires 453.6 to 2110.4 million instructions for QCIF size I-frame and 1175.2 to 5498 million for P-frame. It shows the encoder complexity of H.264/AVC is at least one order more than MPEG-4 base layer Lehtoranta *et al.* [6].

Table 2: "allon" encoder complexity per frame (million)

	I (1st frame)	P (2-50)	Avg (1-50)
Load	690.29	2036.3	2009.4
Store	314.15	845.79	835.16
Unconditional Br	72.60	173.15	171.14
Conditional Br	97.71	190.24	188.39
Integer	924.91	2321.4	2293.5
Floating Point	0.34	0.41	0.40
Total	2100.0	5567.3	5498

4.2. Memory bandwidth

Fig.3 shows that for encoder, the load and store operations take up 29-38% and 12-15%, respectively. Based on the ~2:1 relationship between load and store, associated memory read and memory write circuitries and bus widths can be designed accordingly. It is observed that the load and store operations of inter coding is on the average 3 times more often than that of intra coding. Assuming QCIF 10fps and all load/store are 1 Byte, the memory bandwidth of intra coding is 1.3-6.9GB/sec for reading and 0.53-3.1GB/sec for writing. The bandwidth of inter coding is 4.16-20GB/sec for reading and 1.73-8.4GB/sec for writing.

4.3. Computational complexity

Integer computation dominates the encoder complexity: 44%-50% for intra coding and 42%-45% for inter coding. As expected, floating point operations are almost negligible because H.264/AVC is designed to avoid them. We classifies the MIPS IV integer operations into {add, sub, mult, div, shift, logic, misc} and the breakdown is shown in Fig. 4. It shows add, sub and shift dominate the integer operations. Logic and mult operations take up a small fraction, while div operations are almost negligible. The reference software employs various strategies to convert mult and div to less expensive operations like add and shift. For instance, the coefficients of scaling multiplication in integer transform is integrated into quantization steps and stored as entries in a lookup table. This converts several mult and div operations to one shift only. Similar strategy is applied in inverse integer transform and backward quantization.

Table 1: Simulation configurations

	Intra (III…)		**Inter (IPPPP…)**							
Configuration	1	2	3	4	5	6	7	8	9	10
Label	intra	intra2	alloff	hada-mard	cabac	3refs	meblock	rd	hada-cabac	allon
Hadamard	N	Y	N	Y	N	N	N	N	Y	Y
Ref. Frames	-	-	1	1	1	3	1	1	1	3
Block size	-	-	16x16	16x16	16x16	16x16	All	16x16	16x16	All
B-frames	-	-	N	N	N	N	N	N	N	N
CABAC	N	Y	N	N	Y	N	N	N	Y	Y
RD-optimize	N	Y	N	N	N	N	N	Y	N	Y
Search Range	-	-	16	16	16	16	16	16	16	16

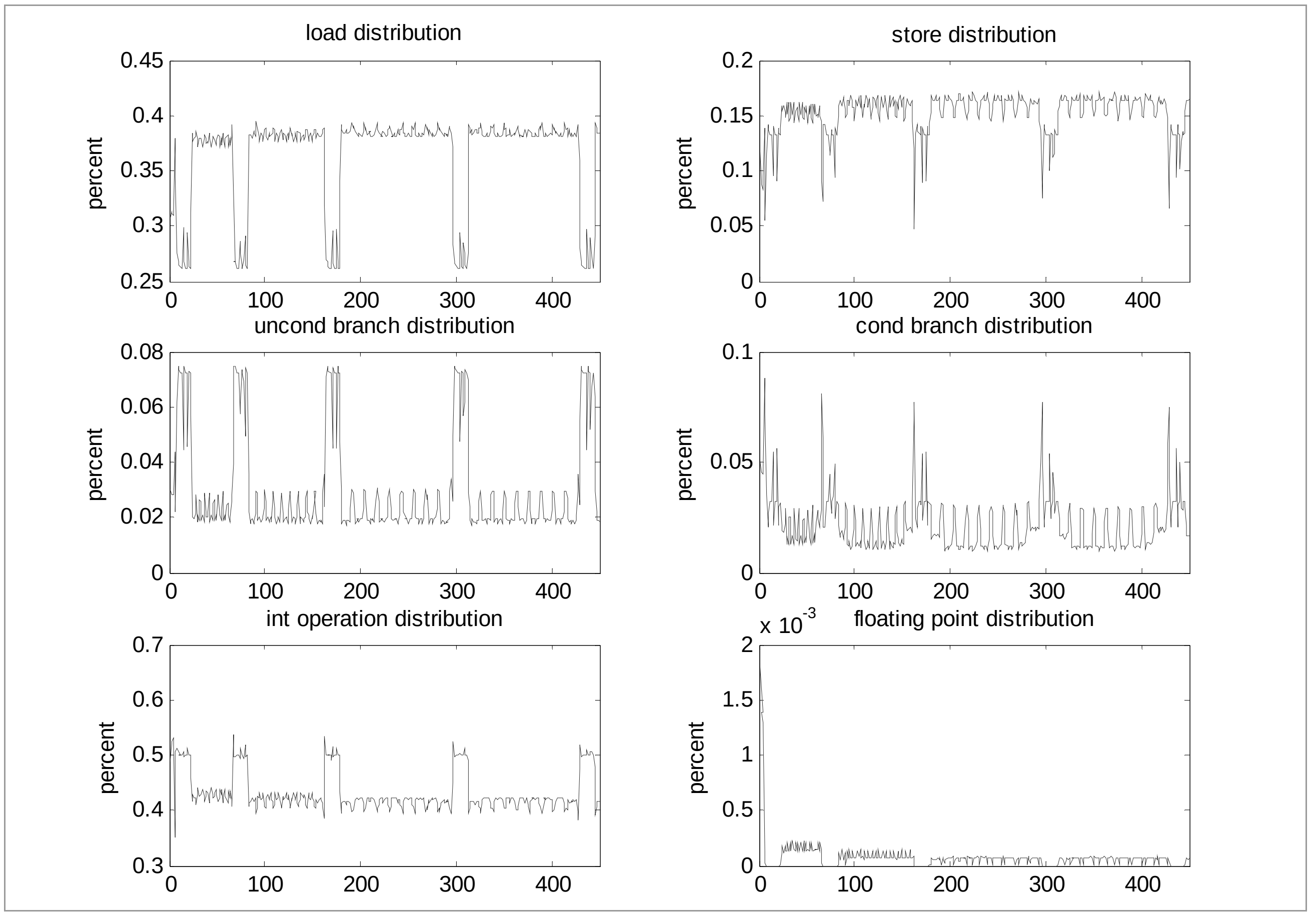

Figure 2: Encoder (config. 6, 3-refernce frames) instruction class distributions in each interval of 20 million instructions. (X-axis indicates interval number)

4.4. Control instructions

For hardware ASIC implementation, branches add difficulty to control logic design since more branches means less regularity in data flow. For software implementation, branches may stall the processor pipeline. Unconditional branch contributes 3.5%-5.7% for intra coding and 2.7%-3.7% for inter coding. Conditional Branch contributes 4%-4.7% for intra coding and 2%-3.6% for inter coding. Although these numbers are not comparable to memory and integer operations, branches could considerably degrade system performance. The measured count is nearly 2G/sec for both branch types when all the tools are switched on.

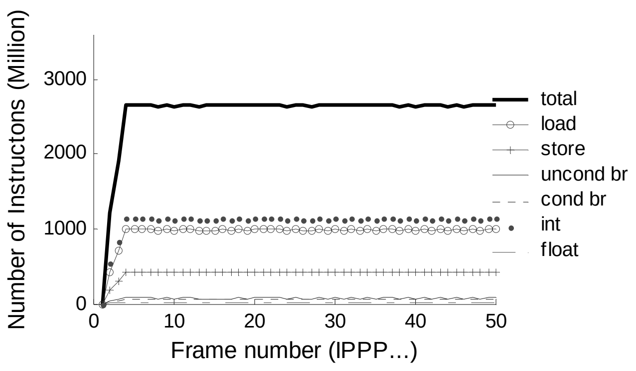

Figure 3: Encoder (config.6) complexity on a frame basis

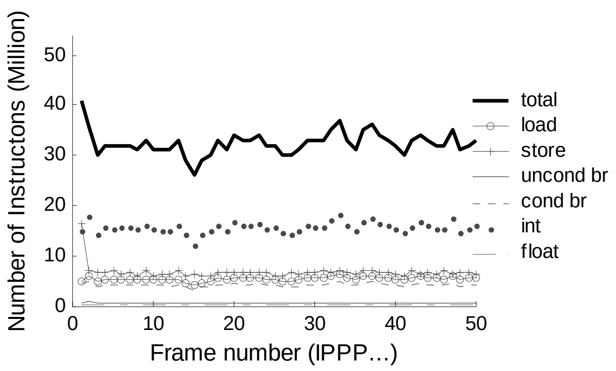

Figure 6: Decoder (config.6) complexity on a frame basis.

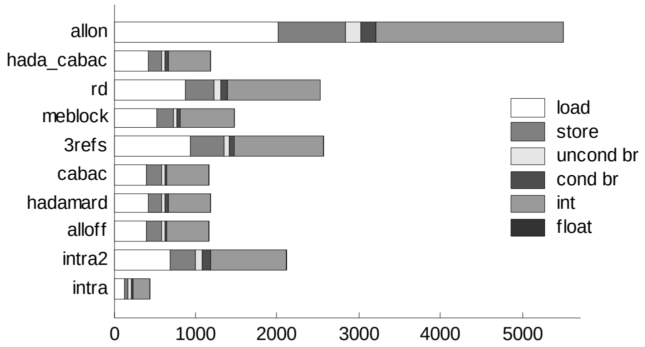

Figure 4: Encoder (all configs) complexity (million) per frame

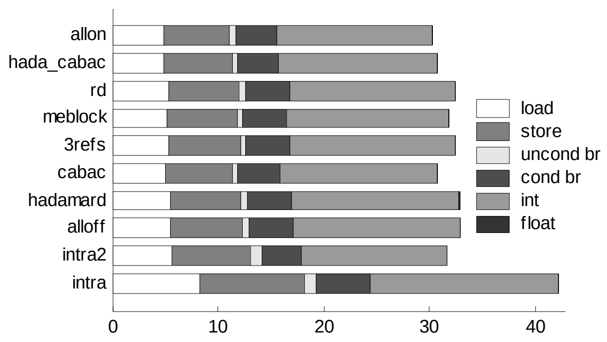

Figure 7: Decoder complexity (million) per frame

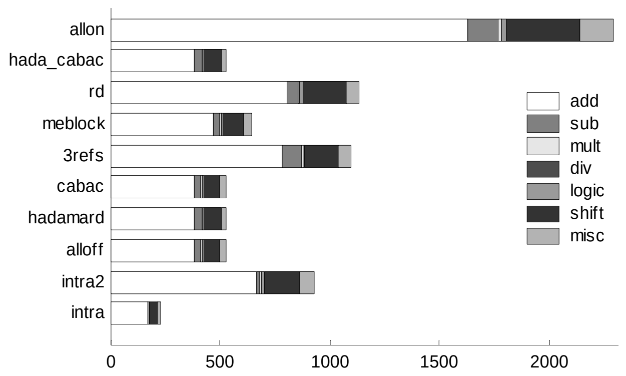

Figure 5: Encoder integer operation complexity (million)

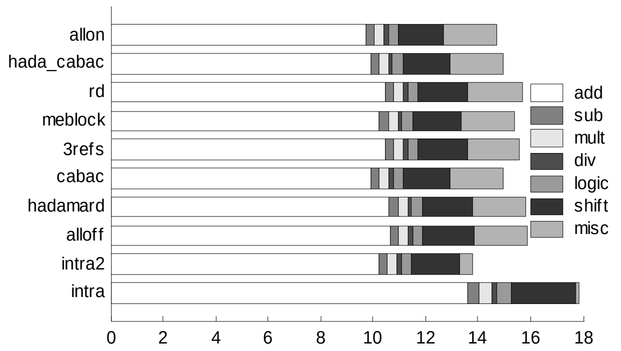

Figure 8: Decoder integer operation complexity (million)

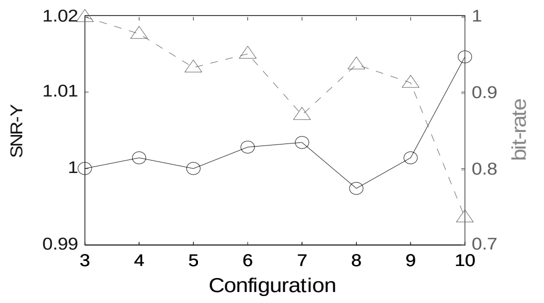

Figure 9: Normalized SNRY and bit-rate

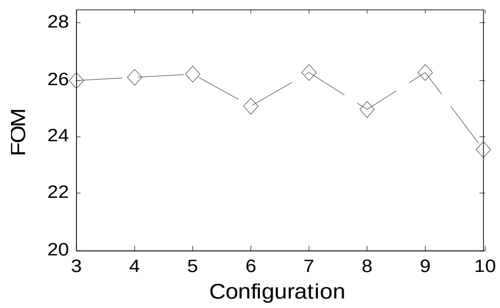

Figure 10: FOM for configuration 3-10

5. COMPLEXITY: DECODER

The encoder shows consistent inter-frame complexity, whereas decoder complexity is more content dependent (20% variations are observed) as shown in Fig 6. Fig 7 shows the decoder complexity does not vary much for different encoder configurations. The average operations per QCIF sized frame is 30-42 million instructions, about 1-2 order of magnitude less than that of the encoder. Integer computation is again the major part, 12.8-17.9 million per I-frame and 14.7-15.8 million per P-frame. Fig 8 gives the breakdown of integer operations. It is worth noting that mult and div take a larger percentage compared to that of encoder. Unlike the encoder, the memory write bandwidth (63-99 MB/sec) is higher than memory read bandwidth (48-82 MB/sec) for the decoder. Unconditional branch takes up a small fraction (0.55-1.1milion) but conditional branch takes up a much larger percentage (3.8-5.1 million).

6. COMPLEXITY, QUALITY, AND BIT RATE TRADE-OFF

The normalized SNR-Y and bit-rate to "alloff" is show in Fig. 9. As for this experiment (low bit-rate, low to moderate scene complexity), CABAC and variable block size improves bit-rate saving at low complexity increase; multiple reference frames improves bit-rate saving but doubles the complexity. When all tools are switched on, 26% bit-rate saving is achieved and SNR-Y is improved by 0.5 dB.

Besides the instruction count complexity measure, it is sometimes desirable to consider the instruction latency and power consumption. For example, power is one of the primary concerns for multimedia processing in handheld devices. To accommodate these concerns, we develop a general complexity metric as follows.

$$Complexity = W_{arith} \cdot C_{arith} + W_{mem} \cdot C_{mem} + W_{control} \cdot C_{control}$$

where *W* is the weight vector and *C* is the measured instruction count vector. The conventional instruction count measure is a special case where all weights are 1.

We further present two weight metrics emphasizing on execution time and power, respectively. These weights have to be specifically chosen for different target processors. As an example in this work, the MIPS R10000 processor [8] is assumed. Table 4 gives the instruction latency and Table 5 gives the function units power consumption for this processor. The power model was presented by *Brook et al.* in their Wattch [9] framework. We configured the parameters according to MIPS R10000 as follows: CMOS technology: 0.35µm, Clock: 195MHz, Vdd: 3.3 V. The integer operations and conditional branches consume integer ALU power, each load and store operation consumes cache/TLB power. Unconditional branch are not counted. Similar complexity metric can be derived easily if a different target processor is to be chosen.

Table 3: Instruction latency as weights

Instruction	Latency	Instruction	Latency
add/sub/logic/set	1	Div	34
mf/mt hi/lo	1	Load	2
shift/lui	1	Store	-
Mult	5	Cond. Br	1

Table 4: Function units power consumption as weights

Component	Function	Power (Watts)
Int ALU 1	add, sub, logic, shift, conditional branch	0.6597
Int ALU 2	add, sub, logic, mult, div	0.6597
L1 data cache	32K, 2 way set-assoc. refill line: 32 byte	4.6444
Data cache TLB	64 entries, full-assoc. page size:4096K	0.5111

To combine the defined complexity, objective visual quality and bit-rate saving into one metric, a figure of merit (FOM) of a particular configuration c relative to the basic configuration "alloff" is introduced and defined as follows:

$$FOM(c) = \alpha \cdot X - \beta \cdot Y - \gamma \cdot Z$$

$$X = \frac{SNRY(c)}{SNRY(alloff)}, Y = \frac{bitrate(c)}{bitrate(alloff)}, Z = \frac{Complexity(c)}{Complexity(alloff)}$$

where α, β, γ are adjustable coefficients. For example, in applications requiring high quality, α should be set large to emphasis quality, whereas in real-time applications, γ should be set small to emphasis complexity saving. α, β, γ may be calculated using "equivalent performance" criteria. For instance, if "1% increase in SNR-Y and 10% increase in bit rate without complexity change" and "1% increase in SNR and 30% increase in complexity without bit rate change" are both considered as good as the original one, then:

$$1.01\alpha - 1.1\beta = \alpha - \beta \quad and \quad 1.01\alpha - 1.3\gamma = \alpha - \gamma$$

One particular solution is $\alpha = 30, \beta = 3, \gamma = 1$. Using instruction count as the complexity measure, along with these parameters, the FOM of all inter-coding configurations in Table 1 is shown in Fig. 10. In this case configuration 9 gives the best FOM. It is important to note that the FOM defined here is not an absolute metric, i.e. it is meaningless to apply it to a single configuration. Instead the FOM is a relative performance measure among different configurations of a single application, or possibly across different applications in the same domain, hence assist system designers to evaluate trade-offs.

7. CONCLUSION

In this paper, a toolset is developed to study the complexity of H.264/AVC encoder and decoder. Analysis shows complexity of inter coding is 3 times intra coding. Memory read bandwidth is 2 times write bandwidth for encoder but less than write bandwidth for decoder. Analysis of integer-ALU instructions shows add, sub, and shift are dominant for both encoder and decoder. These results are particularly useful for resource allocation in hardware implementations. Tool-by-tool analysis shows trade-offs between various configurations. A complexity-quality-bitrate performance metric is presented to examine the relative performance among all the configurations. The metric contains application-dependent coefficients and can be customized to a variety of H.264/AVC application domains. In this work, the configuration with hadamard transform and CABAC achieves highest figure of merit.

8. REFERENCES

[1] D. Burger and T. M. Austin, “The SimpleScaler Tool Set,” Version 2.0. *Computer Architecture News*, page 13-15, June 1997.

[2] “Information technology - Coding of audio-visual objects - Part 10: Advanced video coding,” Final Draft International Standard, ISO/IEC FDIS 14496-10, Dec. 2003.

[3] M. Horowitz, A. Joch, F. Kossentini, and A. Hallapuro, “H.264/AVC Baseline Profile Decoder Complexity Analysis”, *IEEE Transactions on Circuits and Systems for Video Technology*, pp. 704-716, vol. 13, no. 7, Jul. 2003.

[4] V. Lappalainen, A. Hallapuro, T.D. Hamalainen, “Optimization of emerging H.26L video encoder”, *Proc. IEEE Workshop on Signal Processing Systems*, pp. 406-415, September 2001.

[5] V. Lappalainen, A. Hallapuro, T.D. Hamalainen, “Complexity of optimized H.26L video decoder implementation**”**, *IEEE Transactions on Circuits and Systems for Video Technology*, pp.717–725, vol. 13 , no. 7 , July 2003.

[6] O. Lehtoranta, T.D. Hamalainen, “Complexity analysis of spatially scalable MPEG-4 encoder,” *Proc. International Symposium on System-on-Chip*, pp57-60, Nov. 2003.

[7] S. Saponara, C. Blanch, K. Denolf, J. Bormans, “The JVT Advanced Video Coding Standard: Complexity and Performance Analysis on a Tool-by-Tool Basis”**,** *IMEC*, 2003.

[8] MIPS Technologies, “MIPS R10000 Microprocessor User’s Manual, Version 2.0,” MIPS Technologies, 1996.

[9] D. Brooks, V. Tiwari, and M. Martonosi, “Wattch: A Framework for Architectural-Level Power Analysis and Optimizations,” *Proceeding of 27th Annual International Symposium on Computer Architecture*, p.83-94, June 2000.