Journal of VLSI Signal Processing 41, 93–110, 2005

A Software-Hardware Co-Implementation of MPEG-4 Advanced Video Coding (AVC) Decoder with Block Level Pipelining

SHIH-HAO WANG AND WEN-HSIAO PENG
Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan

YUWEN HE
Department of Computer Science and Technology, Tsinghua University, Beijing 100084

GUAN-YI LIN, CHENG-YI LIN, SHIH-CHIEN CHANG, CHUNG-NENG WANG AND TIHAO CHIANG
Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan

Received July 2003; Revised January 2004; Accepted March 2004

Abstract. We present a baseline MPEG-4 Advanced Video Coding (AVC) decoder based on the methodology of joint optimization of software and hardware. The software is first optimized with algorithm improvements for frame buffer management, boundary padding, content-aware inverse transform and context-based entropy decoding. The overall decoding throughput is further enhanced by pipelining the software and the dedicated hardware at macroblock level. The decoder is partitioned into the software and hardware modules according to the target frame rate and complexity profiles. The hardware acceleration modules include motion compensation, inverse transform and loop filtering. By comparing the optimized decoder with the committee reference decoder of Joint Video Team (JVT), the experimental results show improvement on the decoding throughput by 7 to 8 times. On an ARM966 board, the optimized software without hardware acceleration can achieve a decoding rate up to 5.9 frames per second (fps) for QCIF video source. The overall throughput is improved by another 27% to 7.4 fps on the average and up to 11.5 fps for slow motion video sequences. Finally, we provide a theoretical analysis of the ideal performance of the proposed decoder.

Keywords: MPEG-4, advanced video coding (AVC), H.264, joint video team (JVT), software-hardware co-implementation, task partition, MB level pipelining

1. Introduction

Multimedia services over mobile networks are becoming important with the pervasive presence of the Internet and wireless devices. For example, new service such as MMS (Multimedia Messaging Service) over cell phones is booming. The services are usually constrained by limited channel bandwidth. To address the issue, MPEG-4 Advanced Video Coding (AVC) [1] is widely considered for content delivery due to significant improvement in coding efficiency. With high complexity tools in AVC specification, it is challenging to implement AVC encoder and decoder for real-time content delivery. Several AVC codecs [2–4] are optimized for general-purpose processors that have powerful processors, large memory, special media instruction sets and wide buses to facilitate the cost reduction. However, for the portable or mobile devices the design of the optimized AVC modules is constrained by low computational power and small memory spaces. In this paper, we present software/hardware co-implementation to address the performance issue under both constraints.

The design methodology for realizing a multimedia system on a chip can be roughly partitioned into 3 categories covering pure software implementation, pure hardware design and SW/HW joint optimization. The pure software implementation can be found in [5–7], where the existing processors such as RISC CPU or DSP are used as the only platform. The design methodology of pure software implementation has the shortest design cycle and is flexible for a wide range of applications. However, the functionality is limited by the built-in instructions and architecture of the processors. With the pure hardware design, all the modules of a multimedia system are constructed with hardware circuits [8]. The hardware is used for a special-purpose multimedia system to exploit the maximal parallelism and to achieve the best performance simultaneously. The optimization of the dedicated hardware design usually fits the target applications. As to the SW-HW co-implementation, we can achieve both fast development and high performance [9–12]. The SW/HW joint optimization is based on the characteristics of a multimedia system. With the SW/HW co-implementation, the development time of a multimedia system can be significantly reduced by porting the software modules to an existing chip. To enhance the overall system throughput, some modules whose software level optimization still causes the performance bottleneck when running on low power platforms are sped up by dedicated hardware. In addition, the software and hardware tasks are synchronized through parallel processing. To maximize the overall throughput, the issues of task partitioning, software optimization, hardware implementation and task synchronization need to be resolved.

According to the complexity analysis of MPEG-4 AVC decoder, the required computational power is too high for regular RISC processors or DSP to meet the target low bit-rate of multimedia service applications, which usually provide 7.5 QCIF (176×144) frames per second (fps) as the minimum need over mobile network or 3G W-CDMA (Wideband Code Division Multiple Access) [13]. To meet the minimal processing rate of 7.5 fps, we propose an optimized SW/HW architecture for a baseline MPEG-4 AVC decoder. In the design flow, the general-purpose processors control the decoding processes and the dedicated coprocessors conquer the computationally intensive modules. The coprocessors are realized with programmable logic modules to retain the flexibility of circuit design. In addition, a standardized system bus is used to remove the bottleneck of data communications between modules and to allow interoperability. Subsequently, the proposed architecture is evaluated on an ARM966 platform. The results show that the optimized software without hardware acceleration can achieve a decoding rate up to 5.9 frames per second (fps) for the video sequences in QCIF resolution and YCbCr 4:2:0 color format. With the proposed SW/HW design, the throughput is increased by about 27% to reach 7.4 fps on the average and is up to 11.5 fps for slow motion video sequences. To estimate the practical upper bound of our design method, a theoretical analysis is given.

This paper is organized as follows. In Section 2, we present a novel MPEG-4 AVC decoder architecture and analyze the complexity. In Section 3, the Software/Hardware design methodology and architecture are provided. In Section 4, the synchronization and the optimization issues of software and hardware are resolved. In Section 5, the Software/Hardware design results are given and overhead is analyzed to explain the experimental results. Section 6 draws the conclusion.

2. MPEG-4 AVC Decoder

2.1. System Architecture

As shown in Fig. 1, the MPEG-4 AVC decoder can be partitioned into four main modules.

1. Context-adaptive variable length decoding (CAVLD).
2. Inverse quantization and inverse 4×4 Discrete Cosine Transform ($Q^{-1}DCT^{-1}$).
3. Motion compensation (MC).
4. Reconstruction and loop filtering.

The decoding is performed by first parsing the compressed bitstream by CAVLD. After the parsing, the quantized prediction residuals and macroblock (MB) side information including MB type (MB_Type), prediction mode and motion vector difference (MVD) are extracted. The MB_Type information controls the switches S_0 and S_1 in Fig. 1 to determine the prediction type. For an intra MB, the intra prediction is derived from the neighboring pixels without loop filtering. For an inter MB, the inter prediction is derived from motion compensated pixels. The addition of the prediction pixels and the decoded residuals produces the reconstructed frame. After the reconstruction, a loop-filter is applied to remove blocking artifacts and the filtered

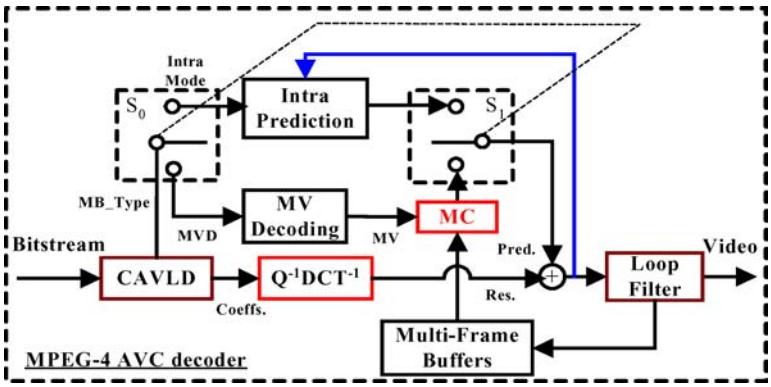


Figure 1. ARM-based architecture of MPEG-4 AVC decoder.

pixels are stored in the frame buffers for advanced reference.

The new MPEG-4 AVC standard contains many powerful tools for enhancing the coding efficiency. For example, new tools for a baseline AVC decoder covering the multiple reference frames (1–5 frames), the variable block sizes for motion search (7 modes for 4×4 to 16×16), the intra prediction, the context adaptive entropy coding, the 4×4 integer inverse DCT transform, the Hadamard transform, and the rate-distortion optimization. With the tools MPEG-4 AVC achieves high coding efficiency but drastically increases the complexity.

2.2. *Complexity Analysis*

To analyze the complexity quantitatively, we have evaluated the AVC and two most popular video standards including MPEG-4 Simple Profile (MPEG-4@SP) and H.263. To evaluate the AVC decoder complexity objectively, we limit the multimedia applications to be compliant with the Baseline Profile at Level 1 as summarized in Table 1. The encoding parameters for complexity comparison are listed in Table 2. Table 3 shows the bit rates and the objective qualities in PSNR for 5 different MPEG test sequences. With an identical bit rate of 64 kbps, MPEG-4@SP has similar quality as that of H.263, and AVC has more than 3 dB gain in PSNR over the 2 previous standards. In addition, to achieve the processing rate up to 7.5 fps of QCIF video, the operation count for each decoder is observed in Table 4. The operation counts with MPEG-4@SP, AVC and H.263 decoder are 39, 218, and 9 million cycles per second, respectively. Based on the operation counts, we can roughly estimate that the execution time of AVC [1] is 6 times compared to that of MPEG-4@SP [14] and 24 times compared to that of H.263 [15].

Table 1. MPEG-4 AVC baseline profile and level 1.

Max MB processing rate	1485
Max picture size	176×144
Max bit rate	64 Kbits/sec
Max # of reference frames	5

3. System Architecture

To enhance the overall throughput of the AVC decoding system, the system architecture is developed based on the characteristics of the decoding modules and data dependency between the modules. Considering the complexity characteristics, we adopt the Software/Hardware (SW/HW) co-implementation design methodology. To resolve the data dependency issue, a time scheduling for MB level pipelining is presented.

3.1. *SW-HW Co-Implementation Design Methodology*

Joint optimization of software and hardware (SW/HW) co-implementation offers a flexible way to balance the performance, cost and complexity. Design methodology of SW/HW co-implementation has been widely adopted for realizing complicated multimedia systems [9–12]. The advantages of such design methodology can be shown from 3 different aspects including

Table 2. The encoding parameters for construcing the test bitstreams of MPEG-4@SP, MPEG-4 AVC and H.263 at 64 kbps.

	Search range	Search algorithm	Format	Resolution	Variable search mode	Multiple reference frame
MPEG-4@SP	±16	Full search	7.5fps QCIF	Half pel	16×16 and 8×8	1
MPEG-4 AVC baseline	±16	Full search	7.5fps QCIF	Quarter pel	7 modes	5
H.263	±15 (max)	Fast motion (default)	7.5fps QCIF	Half pel	16×16	1

Table 3. Rate and distortion performance for MPEG-4@SP, MPEG-4 AVC and H.263.

Sequence	Codec	Bitrate (bps)	PSNR_Y (dB)	PSNR_U (dB)	PSNR_V (dB)
Foreman	MPEG-4@SP [14]	63.49	32.42	37.97	38.17
	H.263 [15]	64.14	32.76	38.17	38.59
	MPEG-4 AVC [1]	60.24	36.02	40.31	41.02
Akiyo	MPEG-4@SP [14]	63.48	41.77	44.64	45.34
	H.263 [15]	64.14	42.86	45.24	46.00
	MPEG-4 AVC [1]	62.69	47.42	48.72	49.17
Mother_daughter	MPEG-4@SP [14]	63.47	39.26	43.81	44.16
	H.263 [15]	63.99	39.87	43.79	44.32
	MPEG-4 AVC [1]	65.25	43.40	45.84	46.36
Coastguard	MPEG-4@SP [14]	63.48	30.97	41.43	42.66
	H.263 [15]	64.02	30.88	40.42	41.99
	MPEG-4 AVC [1]	58.69	32.10	41.37	43.40
Container	MPEG-4@SP [14]	63.49	37.23	41.98	41.66
	H.263 [15]	64.02	37.11	41.85	41.42
	MPEG-4 AVC [1]	60.36	40.61	45.22	45.15

Test machine: Pentium 4-2G.

computational characteristics, design verification and system performance.

Most video coding schemes are composed of both irregular branching operations and regular complex operations. The irregular branching operations with fewer computations are more suitable for software implementation and the regular complex operations can be effectively implemented with dedicated hardware co-processors. For the design verification, software level implementation on the existing platforms such as personal computers (PC) is much easier for debugging and verification. In contrast, hardware level implementation usually takes long design cycle for testing and verification. From the system performance perspective, specialized hardware design with specific functionalities can obtain higher performance than pure software design. Thus, we use the SW/HW joint optimization to retain an optimized operation performance with moderate design efforts.

Table 4. The operation counts for the decoders of MPEG-4@SP, MPEG-4 AVC and H.263.

Sequence\ Codec	MPEG-4 @SP [14] (Mcycles/sec)	MPEG-4 AVC [1] (Mcycles/sec)	H.263 [15] (Mcycles/sec)
Foreman	42.00	237.92	8.89
Akiyo	32.94	192.54	9.73
Mother_ daughter	38.67	218.30	8.82
Coastguard	41.04	238.32	10.55
Container	37.33	200.88	6.98
Average	38.39	217.60	8.89
Ratio	4.27	24.19	1.00

Test machine: Pentium 4-2G.
Mcycles/sec: Million cycles per second.

Several video codec systems [9–12] have been developed using the SW/HW joint optimization approaches. The authors in [9] have developed a H.263 based videophone system with multiple micro-controllers, processors and dedicated hardware. The system is controlled with 2 micro-controllers, which are used for the

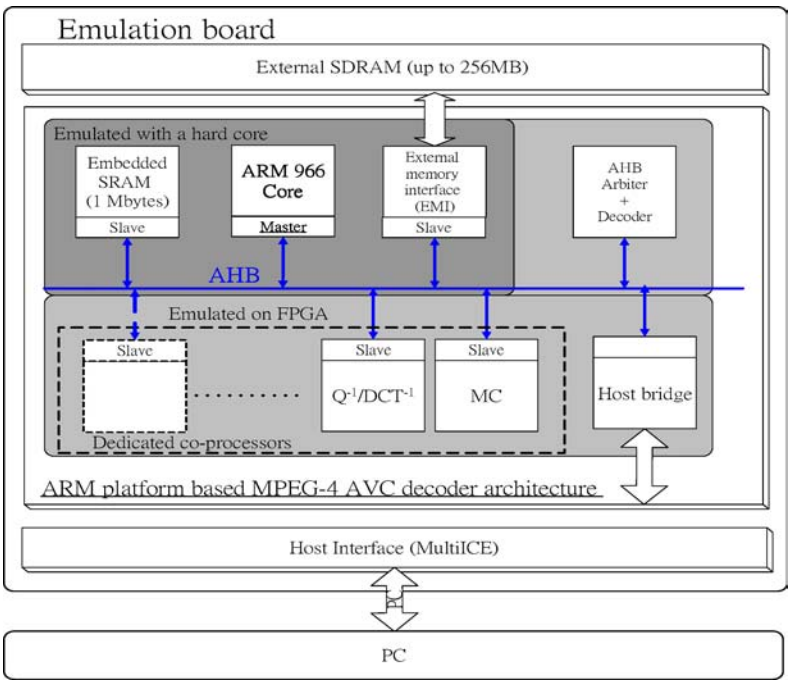


Figure 2. Architecture and emulation environment of ARM platform based MPEG-4 AVC decoder.

MB-level pipelining and the memory management respectively. The computational intensive parts such as motion estimation (ME) or discrete cosine transform (DCT) are processed by the dedicated hardware. Data for different modules are moved via on-chip data and command buses controlled by the micro-controllers. Another similar approach in [11] has proposed a multi-cores architecture for a single chip solution for MPEG-4 streaming application. A RISC processor is embedded for data flow control and a VLIW processor is designed for computationally intensive tasks. In particular, they follow the ARM Master Bus Architecture (AMBA) to implement the system bus for data communication.

Instead of the single chip approaches in [9] and [11], the authors in [10] have developed a scalable MPEG-2 Main Profile and Main Level (MP@ML) encoding scheme. The encoding throughput can be increased by deploying multiple chips for parallel processing. Each chip has 3 layers consisting of the process control, the video processing and the data buffering layers. The process control layer has a RISC processor for video sequence control at the MB level. In the second layer of video processing, multiple dedicated hardware modules and a SIMD processor are used for handling the computationally intensive parts, which mean ME, MC or DCT. To communicate with the other chips, a hardware module is designed for function extension and data sharing.

In addition, the authors in [12] have developed a H.263 codec on an emulation board with one Strong ARM 110 core and 2 FPGA boards. By analyzing the computation power taken in the codec system, the high power-consuming modules including ME and motion compensation (MC) are partitioned and realized with dedicated hardware. The remaining modules such as entropy decoding are done only with software level optimization. The data communication between the processor and the dedicated hardware are processed via Direct Memory Access (DMA). In conclusion, the SW/HW joint optimization has been proven to enhance the performance of the video coding systems [9–12]. Thus, we maximize the throughput of the high complex AVC decoding system based on the SW/HW joint optimization.

3.2. *Proposed Architecture on ARM Development Platform*

Figure 2 depicts the ARM platform based MPEG-4 AVC decoder architecture and the emulation

environment. The system includes an ARM966 CPU, an embedded SRAM, multiple dedicated coprocessors, an external memory interface and a 32-bit wide ARM High-speed Bus (AHB). All functional modules are connected via the AHB bus. The ARM966 CPU acts as a master on the AHB bus and conducts the synchronization among all functional blocks. All the remaining functional blocks that comply with the commands from CPU are slaves. Specifically, the dedicated coprocessors can accelerate computation or reduce memory access for rapid processing. The firmware of the coprocessors and the software for the remainders are stored in the embedded SRAM. In addition to the embedded memory, the decoder also requires external memory for frame buffering with size about 470 kilo-bytes. The access to the external memory is via an external memory interface.

4. Design Methodology

4.1. Software and Hardware Partitioning

Software and hardware task partitioning is critical to the performance enhancement and cost reduction of the entire system. To meet the requirement of decoding QCIF video sequences at 7.5 fps, our system needs a processor running at 218 million cycles per second according to the observation in Section 2. Since our processor runs in 140 MHz, the pure software-level optimization of AVC decoder implementation can only achieve the decoding rate of 4.83 fps. To further increase the decoding throughput to 7.5 fps, we must assign at least 35.7%[1] of workload to dedicated hardware. Thus, the remaining issue of the system architecture is the distribution of the processing tasks to the proper coding modules.

To partition the task, computation characteristics are good criteria to guide the process [9–11]. Table 5 illustrates the kernel operations of each AVC decoding module. Most modules except the $Q^{-1}DCT^{-1}$require a great amount of memory access (MemA). In addition, the MC and loop filtering require intensive arithmetic operations for interpolation and filtering, respectively. The CAVLD needs branch instructions for context-adaptive table switching. Thus, the MC, loop filtering and $Q^{-1}DCT^{-1}$ are suitable for hardware implementation and the CAVLD is suitable for software implementation. Figure 3 shows the decoding profiling in relative execution time for each module. From the Amdahl's law and the observations of Fig. 3, we reduce the computational loads by developing three dedicated coprocessors for the MC, $Q^{-1}DCT^{-1}$ and loop filtering, respectively. The remaining modules are implemented and optimized in software. The ideal upper bound of the decoding frame rate in fps can be approximated when the software and hardware tasks work in parallel with an efficient scheduling. For example, the combined percentage of MC, $Q^{-1}DCT^{-1}$ and loop filtering occupies about 49% for Foreman sequence and 50% for Akiyo sequence. Consequently, we can estimate the ideal upper bound of the decoding rate to be 9.47 fps[2] for Foreman sequence and 9.66 fps[3] for Akiyo sequence.

Table 5. Key operations for AVC decoding modules.

Modules	MC	$Q^{-1}DCT^{-1}$	CAVLD	Loop filtering
Operations	Mul., Add, MemA, Shift	Add, Shift	Branch, MemA.	Add, MemA.

Mul.: multiply, MemA: memory access

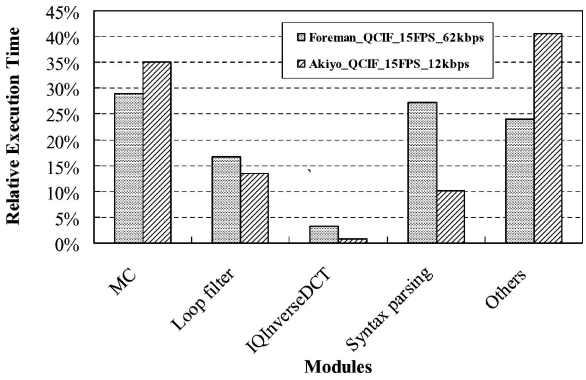


Figure 3. Decoding profiling for AVC on ARM 966 CPU.

4.2. Software and Hardware Synchronization

To save the memory for intermediate data buffers and to maximize the throughput simultaneously, module synchronization is required for SW/HW joint optimization. Module synchronization is realized as a scheduling approach for MB level pipelining as shown in Fig. 4. The scheduling is based on data dependency and workload distribution. The subscript n denotes the MB index. Basically, a MB is decoded through the three stages including (1) CAVLD; (2) $Q^{-1}DCT^{-1}$ and MC; (3) reconstruction and loop filtering.

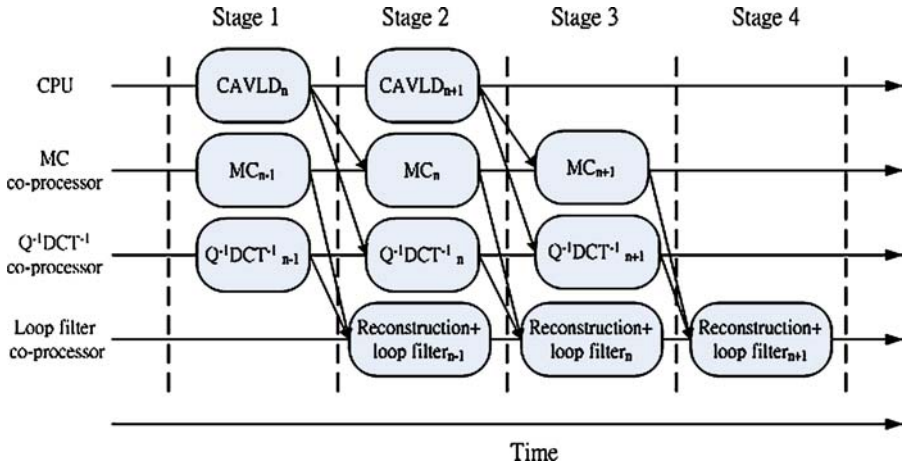


Figure 4. Illustration of proposed MB level pipeline.

Figure 5 shows the flow chart for synchronization between CPU and the three coprocessors in MB level. For synchronization, each stage of decoding processing is initialized as follows.

1. Send the data to the MC coprocessor for motion compensation of the n-th MB.
2. Send the data to the $Q^{-1}DCT^{-1}$ coprocessor for $Q^{-1}DCT^{-1}$ of the n-th MB.
3. Send the data to the LF coprocessor for loop filtering of the $(n-1)$-th MB.
4. Proceed to decode the $(n+1)$-th MB header and coefficients on CPU.
5. Check the ready signals from the three coprocessors.
6. If the data in the three coprocessors are both ready, store the data for loop filtering to reconstruct the n-th MB in the next stage, and receive the filtered data to the $(n-1)$-th MB. Otherwise, go to Stage 5.

4.3. Software Optimization

To retain the flexibility and portability across different platforms, we adopt algorithm level optimization for the software. The main optimization methods are introduced in the following.

4.3.1. Frame Buffer Management. Three frame buffer management methods are adopted to save execution time of memory copy.

1. The frame copy is done by swapping the pointers of frame buffers.
2. The motion compensated frame or intra prediction pixels are stored directly in the decoded frame buffer to avoid memory copy.
3. The decoded residuals are stored in the frame buffers with 4-byte alignment to facilitate access on 32-bit ARM CPU architecture.

4.3.2. Boundary Padding for Motion Compensation. Since unconstrained motion vector and the 6-tap filter [1, −5, 20, 20, −5, 1] are used for MC with half-pel interpolation, the locations of the reference pixels may exceed the frame boundaries. For boundary checking and pixel padding, on-the-fly pixel padding is costly and can be avoided by padding the reference frame before MC. For AVC specification of baseline profile at level 1.0, legal search range of ± 64 is supported, but the memory usage is over 3 times of the original memory size for QCIF resolution.[4] To save memory spaces our solution is to pad $(16+2)$ points to the left and upper edges and $(16+3)$ points to the right and bottom edges. This is because that the data in the padded area is the same. When the motion predictors exceed the padding range, we will map the destination points inside the range that has the identical content. In addition to padding, we also speed up the interpolation by replacing the filter tap multiplication with a table lookup technique.

4.3.3. Content-Aware $Q^{-1}DCT^{-1}$. We observe that most DCT coefficients of motion compensated residuals are quantized to zeros at low bit rate coding. Table 6 shows the percentage of the blocks with all zero

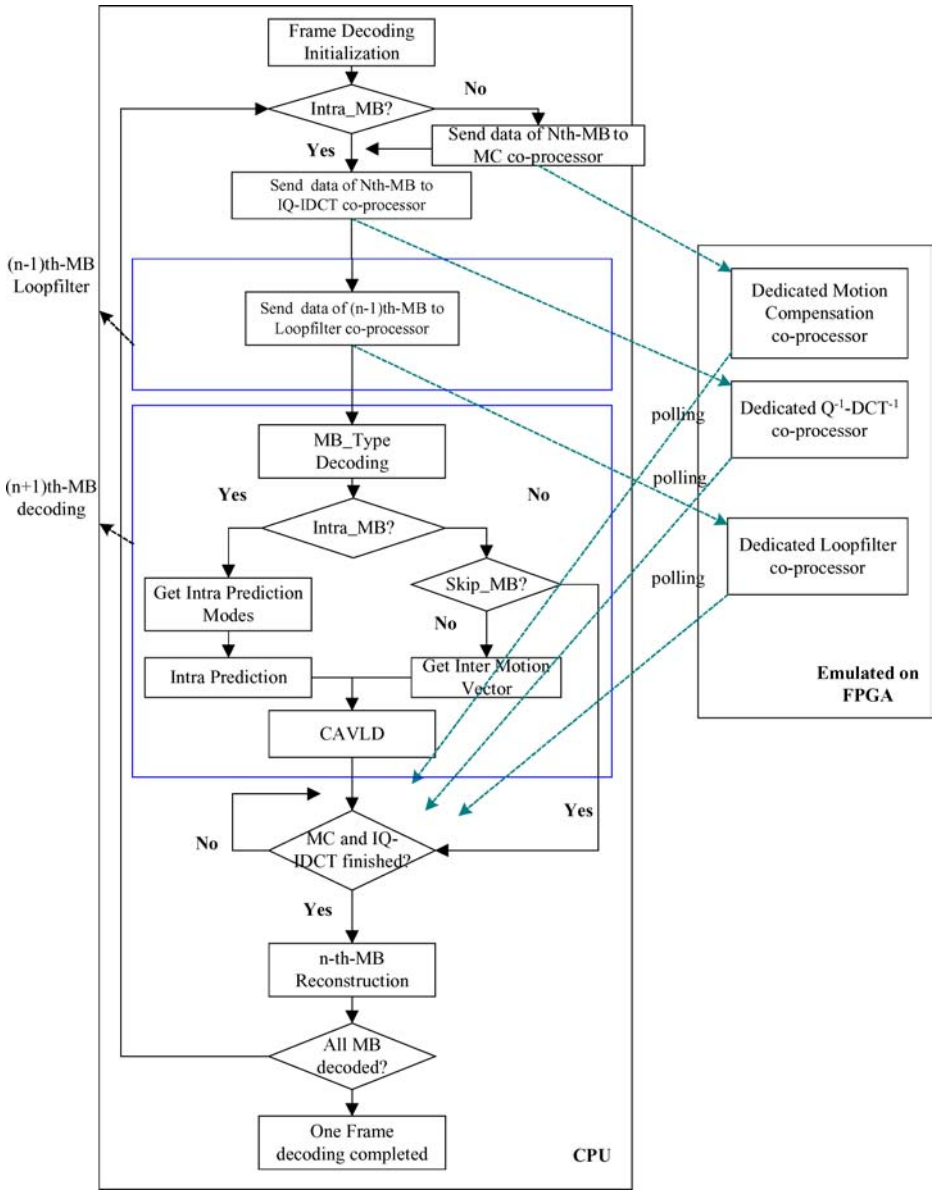


Figure 5. Software flow for MB level pipeline.

coefficients in fast motion and slow motion sequences. For fast motion sequences, about 25% of DCT blocks are all zero cases. For slow motion sequences, the percentage is increased to 47%. In addition, a large percentage of DCT blocks have only one DC coefficient. Thus, we can classify DCT blocks into three categories and conduct $Q^{-1}DCT^{-1}$ differently.

Table 6. Percentage of all zero blocks at 64 kilobits/sec.

Sequence	Fast motion Foreman	Show motion Mother-Daughter
All zero blocks	24.8%	47%

1. For the blocks with nonzero AC coefficients, we perform normal $Q^{-1}DCT^{-1}$.
2. For the blocks with only nonzero DC coefficient, we only perform inverse DC transform.
3. For blocks of all zero coefficients, we just skip the transform.

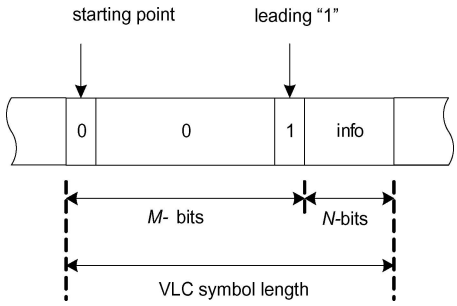


Figure 6. The composition of a new symbol in the bitstream.

4.3.4. A Fast CAVLD Using a Table Lookup Technique. MPEG-4 AVC baseline profile uses many coding tables for CAVLD [16]. Each context refers to a different table for decoding. Currently, reference software uses exhaustive search for symbol decoding [1]. The symbol matching is conducted in a bit-by-bit manner. As a result, decoding each symbol takes many branching and comparison instructions in CAVLD module, which reduces the overall performance.

To speed up CAVLD, we replace the branching and comparison instructions with a table lookup technique. Figure 6 shows that each VLC symbol in AVC bitstreams is composed of leading zeros (M bits) and non-leading zero part (N bits). To speed up the symbol decoding, we organize new VLD tables to avoid branching and comparison instructions. With the novel table lookup method, the symbol decoding is identical to the table indexing. In the new VLD tables, each decoded symbol can be indexed by the length of leading zeros and the value of non-zero part. For example, Table 7 is used to decode the leading zeros of coming symbol. Each entry of Table 7 maps the input data of length k to the corresponding output index, which indicates the length of leading zeros. To balance performance and memory, we choose k to be 8, which takes one byte. After checking leading zeros parts, we will fetch N bits at the same time to decode the non-zero part (or info part in Fig. 6) by table lookup. Since non-zero part occupies smaller than 4 bits in AVC syntax, each entry of the new lookup table takes 8 bits. Thus, the lookup table for symbols with leading zeros (M bits) and non-leading zero part (N bits) has a total size of $M \times 2^N$ bytes.

Table 7. The "leading one" indexing table and its relative return values.

Return value	Symbol	Range of value
1	1xxxxxxxx	128–255
2	01xxxxxxx	64–127
3	001xxxxxx	32–63
4	0001xxxxx	16–31
5	00001xxxx	8–15
6	000001xxx	4–7
7	0000001xx	2–3
8	00000001	1
Jump to the next byte	00000000	0

For decoding a symbol in Fig. 7, our proposed approach needs to look up the table of leading zeros at most twice. The longest symbol length is 16 bits. Consequently, 2 byte-wise operations can fully decode leading zeros and one operation is required for completing the non-zero part. With the specified tables, the total amount of the extra memory required is less than 4 kilo-bytes, which may be proper by considering the increased throughputs. As compared to the decoding rates of [17], we can achieve 37% savings for CAVLC and 40% savings for UVLC (or called Exp-Golomb) on a Pentium-IV 2.0 GHz desktop as shown in Fig. 8. For processing the bitstreams with 300 kbps, the overall throughput is improved by 7.12% on a Pentium-IV

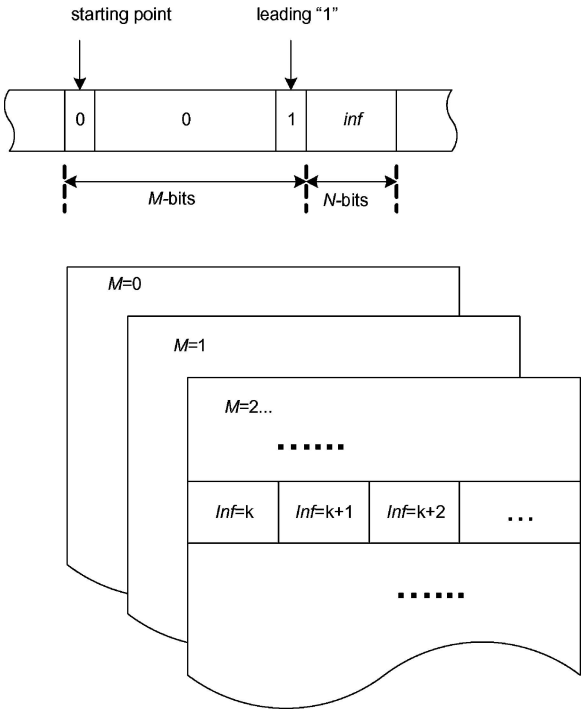


Figure 7. CAVLD symbol decoding using a table lookup technique.

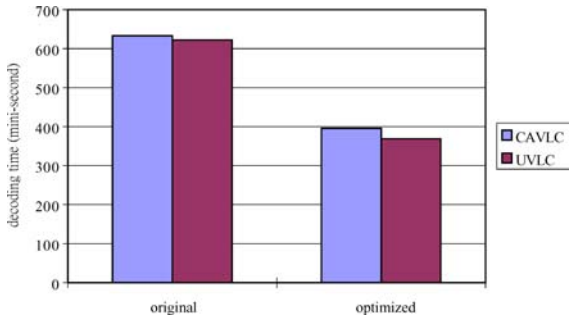


Figure 8. Performance evaluation for CAVLC and UVLC decoding.

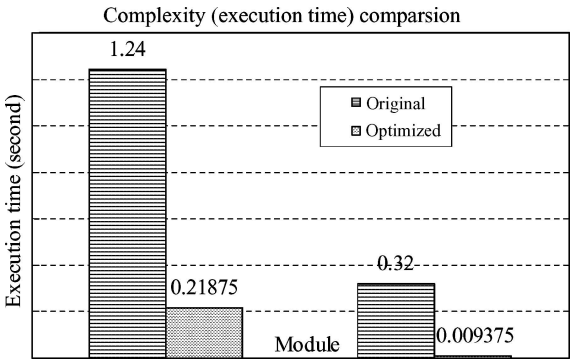


Figure 9. Execution time comparison of MPEG-4 AVC decoders with and without optimization.

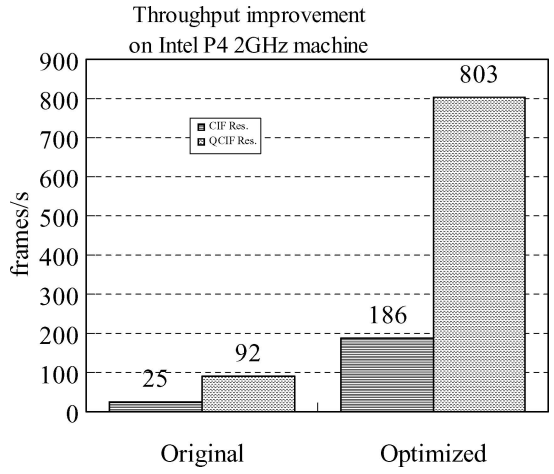


Figure 10. Throughput comparison on the decoding frame rates for MPEG-4 AVC decoders with and without optimization.

2.0 GHz desktop and by 8.3% on the proposed ARM platform.

In our decoding system, the decoding throughput and program code size are both optimized. After our optimization, the code size of baseline decoder is reduced from 244 kilo-bytes (KB) (JM 6.1 [1]) to current 92 KB, which presents 62.3% reduction. As shown in Fig. 9, the overall throughput of CIF/QCIF resolution video sequences is enhanced from 25/92 fps to current 186/803 fps, which shows a speedup of 7.44/8.72 times, respectively. To give a complexity analysis of each module, the operation counts in Fig. 10 are also greatly reduced with more than 80% time saving as compared to the original decoder [1].

4.4. Dedicated Coprocessor Design

With the MB level scheduling, the minimum granularity of a task for the dedicated coprocessors is one MB. In designing the hardware, we hope to build an area efficient architecture under the throughput requirement of decoding QCIF video sequences at 7.5 fps.

4.4.1. MB Based MC. To speed up the interpolation prior to MC, we design a dedicated coprocessor in Fig. 11. For a MB interpolation, the coprocessor uses a local memory to store 1500 integer pixels. It also has two interpolation engines for luminance and chrominance component respectively. With the two interpolators, interpolation process can be sped up by parallel processing. Each engine consists of multiple multipliers and accumulators. The multiplier is implemented in a hardwired manner to maximize performance. Within each MB requiring 2-D interpolation, the intermediate results (output) of row filtering in each engine is fed back to conduct the filtering of another columns.

AVC uses variable block size motion compensation with smallest size of 4×4. Thus, our interpolation engine is designed for a 4×4 block. The interpolation of each MB takes 16 iterations. In the worse case, our design takes 1280 cycles to interpolate one MB. As operating at 10 MHz, the throughput for each MB is 7812 MBs per second (MBs/sec) exceeding the requirements in Table 1.

4.4.2. MB Based $Q^{-1}DCT^{-1}$. The $Q^{-1}DCT^{-1}$ module involves two major tasks. The first task is to store a MB into a local memory with size 3 kilo-bits. The second task is the 4×4 integer inverse DCT, which consists of two multiplication-free transforms and one transpose operation. The IQ is done with a hardwired multiplier. In our implementation, we integrate three different types of inverse transform for luminance 4×4

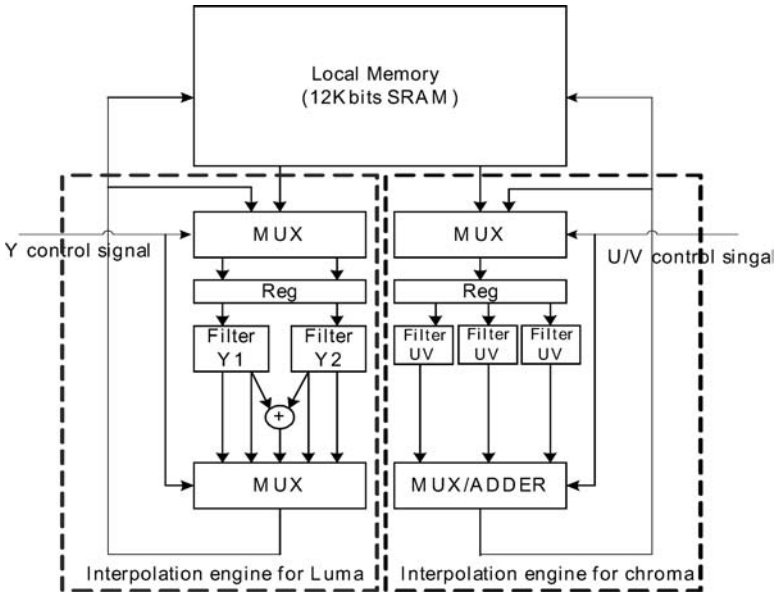


Figure 11. Interpolation architecture for quarter-pel motion compensation.

DC block, chrominance 2×2 DC block and luminance 4×4 blocks, respectively. In the worse case, the $Q^{-1}DCT^{-1}$ takes 210 cycles for one MB. When operating at 10 MHz, the MB throughput is 47619 MBs/sec, which is over the requirement of the AVC decoder in Table 1.

4.4.3. MB Based Loop Filtering. The main idea of MB based loop filtering design is to modify the original frame based loop filtering to fit our design optimization. The MB based architecture introduces 2 advantages including less storage memory and no macroblock dependency in a frame for deeper pipelining. From the viewpoint of memory usage, the original method in the reference software of JM 6.1 needs two buffers of frame size to store the un-loop filtered and loop filtered data. In the proposed MB loop filtering, at most 2 rows of MBs (current and top row) loop filtered data are needed to be buffered. Because the current MB will need only upper and left MB data for loop filtering. The other rows of loop filtered data can overwrite the unused data that will not be used again. Thus, MB based loop filtering can achieve almost 50% reduction of the redundant memory for QCIF or CIF resolution video frames. From the viewpoint of parallel processing, the proposed scheme deepens the pipeline stages according to the synchronization scheduling in Section 4.2 and Fig. 4. Thus MB based loop filtering is well scheduled to work with the MB based MC and $Q^{-1}DCT^{-1}$ modules in FPGA modules.

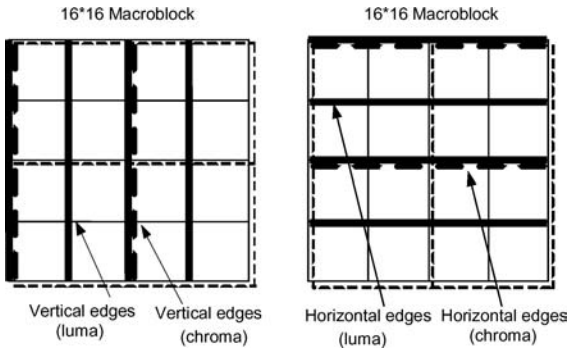


Figure 12. Loop filtering on 4×4 block edges in a macroblock.

Two-dimensional (2-D) loop filtering of the AVC decoding system is processed in a separable manner. To execute the 2-D loop filtering, one-dimensional (1-D) filtering is applied horizontally and then vertically to the handling pixels. As shown in Fig. 12, each 1-D filtering involves at most 16 iterations of edge filtering for the 4×4 blocks. In the worst case, the loop filtering takes 480 cycles to complete the operations in one MB including luminance and chrominance components. When operating at 10 MHz, the MB throughput for the loop filtering is 20833 MBs/sec, which fits the requirement of the AVC decoder in Table 1. In addition,

Table 8. Summary of the dedicated coprocessors.

Modules	MC	LF	$Q^{-1}DCT^{-1}$
RAM size (Kbits)	12	3.84	3
Gate count	11172	6156	11489
Throughput (MBs/sec) (Running in 10 MHz)	7812	20833	47619
Longest delay to run a MB (cycles)	1280	480	210

we need to reserve 3.84 kilo-bits local memory space for loop filtering coprocessor.

To simplify our hardware design and reduce the cost, we replace 2-D interpolation for MC by applying 1-D interpolation filter twice. Although the 1-D interpolation incurs longer delay, it is still fast enough to meet the target frame rate of 7.5 fps.

The total gate counts required for all dedicated coprocessors are 28826. The MC coprocessor contributes 11,172 gates, $Q^{-1}DCT^{-1}$ coprocessor occupies 11,489 gates, and loop filtering coprocessor requires 6,156 gates. Table 8 summarizes the design parameters for each coprocessor.

5. Experimental Results

5.1. Performance Evaluation of Optimized Software

To show the performance of our optimized software, we use the work in [4] as the baseline system for comparison. In [4], a moderately optimized AVC decoder adopts algorithm level optimization techniques and a highly optimized version using Intel MMX technology. In our comparison, all simulations are done on a Pentium III processor with 500 MHz CPU and our optimized software is built from the reference software of version JM6.1. Since the baseline system uses different machines and different reference software, we scale our decoding throughput to match the configuration of baseline system in [4]. Table 9 shows that there are only minor updates from the reference software of version JM5.0h to that of version JM6.1, which indicates that the two versions have similar complexity. In addition, we scale our decoding throughput by a factor of 4/5 for a fair comparison with the results in [4], which are evaluated on a Pentium III processor with a 400 MHz CPU [4]. Because the encoding information provided is insufficient in [4], we assume that all the baseline tools including rate-distortion optimization, Hadamard transform, and intra prediction were enabled in [4]. Thus, we then encode the test bitstreams on the same basis as illustrated in Table 10. The throughput ratio over the decoding frame rate of [4] is defined by

Table 9. Updates for AVC codec from JM5.0h to JM6.1.

New features	• Slice header and parameter sets (except POC) (JM5.0h)
	• Chroma loop filtering update (JM 6.0)
	• SEI decoding re-enabled and new messages added (JM 6.1)
	• Weighted Prediction Updates (JM 6.1)
	• POC updated (JM 6.1)
	• CABAC slice initializations (JM 6.1)
Removed feature	• ABT (JM5.0h)

$$Ratio = \left(\frac{4}{5} fps_{\text{proposed}}\right) \Big/ \left(fps_{\text{ref[4]}}\right) \qquad (1)$$

In Table 11, both the medium optimized version (M.O.) and highly optimized version (H.O.) in [4] are compared. The results show that we get a ratio of 2.82–3.52, which means the proposed AVC decoder is 2.8 times faster than the M.O. [4] on a similar platform. The results in Table 11 show that we can get a ratio of 2.16–2.67, which indicates the proposed decoder has double throughput over that by H.O. decoder [4].

5.2. Performance Evaluation of Proposed Embedded System

Our decoding system is emulated with the configuration as described in Fig. 2. The ARM966 CPU (system clock is set as 140 MHz), embedded SRAM (1 Mbytes) and external memory interface are replaced with a dedicated hard core. The other parts of the decoding system including the dedicated coprocessors are implemented on a FPGA module (running at 10 MHz). In addition, the AHB bus is running at 33 MHz. No cache is enabled in this system.

Table 12 lists the decoding throughput in fps on ARM966 CPU. If only the optimized software was used, the decoding speed ranges from 3.4 to 9.4 fps with an average of 5.9 fps. With the aid of the coprocessors, the throughput can be further increased by 17–38%. The overall throughput ranges from 4.4 to 11.5 fps with the average value of 7.4 fps.

Table 10. Test conditions for Lapplainen's work [4] and the proposed.

	Lapplainen's [4] (Moderately Optimized)	Lapplainen's [4] (Highly Optimized)	The proposed
Version	JM 5.2		JM 6.1
Test platform	Pentium III 400 MHz		Pentium III 500 MHz
Sequence format	QCIF		
MMX support	No	Yes	No
Compiler	Visual C++ 6.0	Intel C compiler 4.0	Visual C++ 6.0
Encoding parameter	• Fixed QP • Reference frame rate: 30fps • Encoded frame rate: 10fps • 1 reference frame • 7 modes of search block		• Fixed QP • Reference frame rate: 30 fps • Encoded frame rate: 10 fps • 5 reference frame • 7 modes of search block • Search range of ±16 • Hadamard transform • Intra prediction • Rate Distortion • optimization • GOV: 1I/99P

Table 11. Performance evaluation for Lapplainen's work (Moderately optimized. and highly optimized version) [4] with the proposed.

Sequence (QCIF)	Version	Bit rate (kbps)	PSNR_Y (dB)	PSNR_U (dB)	PSNR_V (dB)	Decoding rate (fps)	Throughput ratio[b]
Foreman	M.O. [4]	24	28.5	37.6	37.7	66.4	2.87
	H. O. [4]	24	28.5	37.6	37.7	88.1	2.16
	Proposed[a]	26.93	30.6	37.7	37.9	238.1	1
Akiyo	M.O. [4]	24	40.9	44.2	44.9	94.7	3.52
	H. O. [4]	24	40.9	44.2	44.9	125.0	2.67
	Proposed[a]	25.49	40.9	43.0	43.8	416.7	1
Mother_Daughter	M.O. [4]	24	37.2	42.6	43.1	81.6	2.97
	H. O. [4]	24	37.2	42.6	43.1	101.5	2.39
	Proposed[a]	25.24	37.4	41.4	42.2	303.0	1

[a]Size = QCIF, Group of Picture = 1I + 149P, QP = I(30)P(31), frame rate = 15 fps Reference frames = 5.
[b]Ratio = $(\frac{4}{5} \cdot fps_{\text{proposed}})/(fps_{\text{ref[4]}})$ (unit: fps).

In Table 12, we observe that the decoding throughput is sequence dependent. Our decoder performs better for slow motion sequences, which have more zero DCT blocks and higher probability of using integer-pixel MC. Therefore, the decoding rate is increased due to the less interpolation of MC and less computation of $Q^{-1}DCT^{-1}$ for slow motion sequences. The improvement ratio using coprocessors for decoding slow motion sequences becomes minor, which is consistent with the observations of Amdahl's law and the results in Table 12.

5.3. Decoding Throughput Analysis with Overhead

In Section 4.1, we approximate the ideal decoding frame rates using the proposed synchronization scheme. Since the operation counts of the decoder are

Table 12. Performance of our prototype decoder.

QCIF	Bit rate (kbits/s)	PSNR_Y (dB)	SW only (fps)	SW + HW (fps)	Throughput speedup (%)
Container	13.15	34.18	8.56	9.98	17
Akiyo	8.67	36.50	9.36	11.50	22
Coastguard	58.69	32.10	3.40	4.40	29
Mother_Daughter	14.43	35.55	5.55	7.68	38
Foreman	42.47	34.00	3.53	4.54	29
Silent	25.99	34.06	6.03	7.29	21
Mobile_calendar	58.88	28.40	3.96	5.20	31
Table tennis	38.83	34.00	5.19	6.59	27
Stefan	54.27	26.23	4.07	5.30	30
Hall_monitor	16.58	35.50	8.91	11.11	25

Size = QCIF, Group of Picture = 1I+74P, frame rate = 7.5 fps.
Reference frames = 5, 7 modes of block size.

evaluated on a desktop PC, we may expect more operation counts on ARM platforms. Because the ARM platforms do not support powerful resources, such as large capacity of cache size (512 Kbytes L2 cache), higher frequency DDR ram (333 MHz), etc, the operation data in Section 4.1 may not reflect the real performance of our proposed platform. Therefore, we re-evaluate the data to study a more realistic scenario for an ARM platform.

In Section 5.2, the pure software decoding frame rate on ARM processor is 5.9 fps on the average, which leads to that 175 Mcycles/sec[5] is required to achieve the throughput of 7.5 fps with QCIF resolution on the ARM platform. Due to the limited resources on the ARM platform and lower clock frequency, it is difficult to achieve this target using the pure software decoder. By analyzing the SW/HW partitioning scheme and the weightings of dedicated hardware modules for the MC, the inverse DCT and the loop filtering in Fig. 3, the ideal upper bound is 11.8 fps.[6] The practical decoding rate of 7.4 fps is smaller than that of ideal case due to the overhead for synchronization and data transfer.

Synchronization introduces additional memory access. In our current implementation, the input data of each coprocessor always come from the CPU. Therefore, the CPU needs extra memory space for data transport among different coprocessors. In addition, the memory movement incurs great penalty during the preparation of input data for each coprocessor. Thus, the synchronization overhead causes the performance degradation.

The other overhead is data movement. In the ideal case, we assume the time for data movement can be ignored. Observing the scheme in Fig. 13(a), we assume that the tasks in each coprocessor and CPU can be fully parallelized in each pipeline stage, and syntax parsing can cover all the running time slots of coprocessors. In practice, the data is required to sequentially move from CPU to each coprocessor. In addition, for rapid and efficient transmission via bus we need extra time to pack the data. Even with the data packing, the buses may conflict as in a more realistic scenario depicted in Fig. 13(b), which takes some CPU cycles. With the overhead for data movement, duration for each pipeline stage is increased. As a result, the overall throughput is decreased. Table 13 shows the amount of data transferring between CPU and coprocessors for a MB computation. To complete a MB processing, CPU has to transfer 4.97 kilo-bytes data in total. Consequently, the total data transferred could be up to 492.3 kilo-bytes for decoding a QCIF frame. Thus, the bandwidth of system bus and the overhead for synchronization set an upper bound for the decoding throughput. The best throughput achieved on the consideration of both the overhead for synchronization and data transfer can be estimated by

$$fps_{\text{ideal}} = fps_{\text{SW}}/(1 - R_{\text{HW}}) \tag{2}$$

$$fps_{\text{practical}} = \frac{fps_{\text{SW}}}{\{(1 - R_{\text{HW}}) * T_{\text{SW}} + T_{\text{data\_transfer}} + T_{\text{synchronization}})\}/T_{\text{SW}}} \tag{3}$$

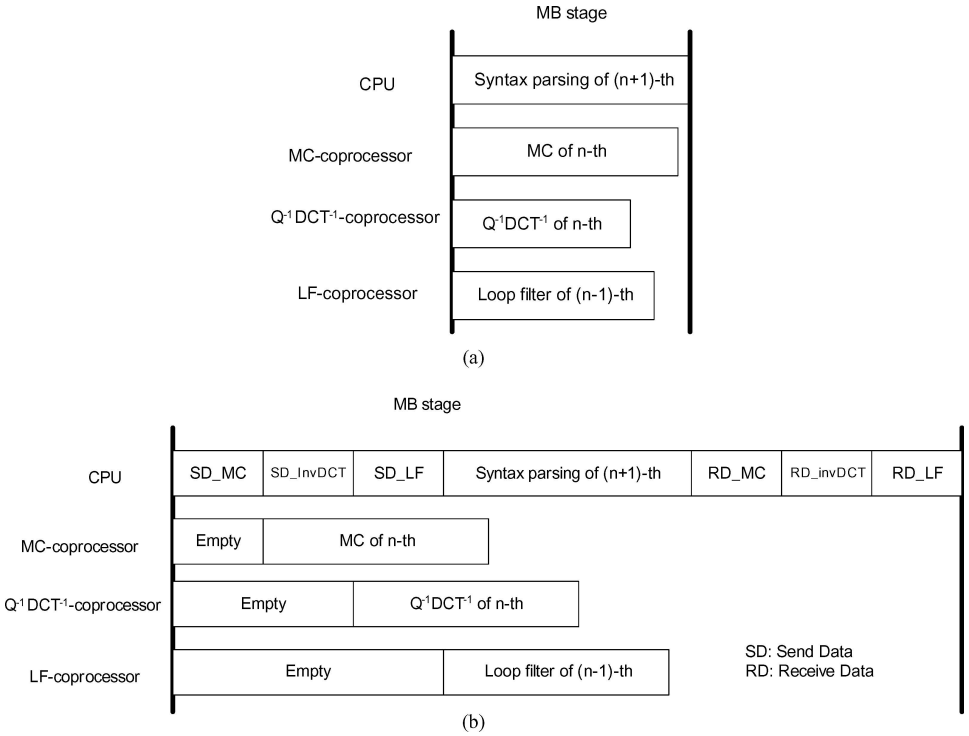


Figure 13. Time scheduling with the proposed synchronization scheme. (a) ideal case (b) practical case.

Where $fps_{\text{ideal}}$is the ideal decoding frame rate and $fps_{\text{practical}}$ is the decoding frame rate with overhead. R_{HW} means the ratio for dedicated hardware. T_{SW} is the total time used for running software level AVC decoder and $T_{\text{data\_transfer}}$ is additional overhead for data transferring. $T_{\text{synchronization}}$ indicates additional overhead for synchronization.

Table 13. Maximum amount of sent or received data for MB based MC, $Q^{-1}DCT^{-1}$ and loop filtering.

Modules	Parameter	Data sending (bytes)	Data receiving (bytes)
MC	Luma	1296	256
	Chroma	256	128
	MV info	64	–
	Total	1616	384
Loop Filtering	Luma and boundary	384	384
	Chroma and boundary	192	192
	Edge info and QP	4	–
	Total	580	576
$Q^{-1}DCT^{-1}$	Luma	1296	256
	Chroma	256	128
	Info and QP	8	–
	Total	1560	384
Total		3756	1344

The cost of extra memory movement for synchronization is 5%, and the term of R_{HW} is almost 50% as demonstrated in Fig. 3. Since the AHB is operated at 33 MHz for 32-bit data, we can approximate that the data transferring via the AHB bus takes 3.64 milliseconds for each frame (492.3 kilo bytes/33 $\times$ 4 mega bytes per second). In the preceding sections, we get the decoding frame rate to be 5.9 fps on the average. The results show that about 1/6 seconds are spent for decoding a frame in SW level. To calculate the practical decoding frame rate according to Eq. (3), we can get

$$fps_{\text{practical}} = \frac{1}{(1-50\%)\cdot\frac{1}{6}+\left(\frac{1}{6}\cdot 5\%\right)+\left(\frac{492.3}{33*4*1024}\cdot h\right)}$$

where h is large than or equal to 5.9 (improved decoding frame rate will be greater than 5.9 fps of pure SW decoding). Thus, we can obtain the practical decoding frame rate to be 8.8 fps that is closer to the experimental results of average 7.4 fps. According to the above

analysis, we can reasonably explain the non-ideal experimental results.

6. Conclusions

In this paper, we have presented a MB level pipelining architecture for a MPEG-4 AVC baseline decoder based on both SH/HW joint optimization design methodology and circuit design principles. In the aspect of software, our work is highly optimized for the ARM architecture with innovative algorithms for frame buffer management, boundary padding, content-aware inverse transform and context-based entropy decoding. The algorithms are specifically designed for the AVC decoder to achieve a speedup by 7 to 8 times. In addition, the code size is as small as 92 kilo-bytes that can easily fit into an existing mobile device. As for the hardware aspect, we proposed a MB level pipelining based on complexity analysis to yield the best partitioning. The scheduling approach considers the tradeoff between efficiencies and flexibility. We implemented three dedicated coprocessors for the MC, $Q^{-1}DCT^{-1}$, and loop filtering modules with only 28,826 gates. With such small additional chip area, the three coprocessors can improve the performance by 27 % that justifies the additional complexity.

In conclusion, our design has shown a highly efficient and cost effective implementation of an AVC decoder. The AVC specific optimization demonstrates that MPEG-4 AVC standard may be applicable for the next generation mobile multimedia communications using a platform-based design methodology.

Notes

1. $\frac{217.6(\text{Mcycles})-140(\text{Mcycles})}{217.6(\text{Mcycles})} = 35.7\%$.
2. $4.83 \cdot \frac{1}{100\%-49\%} = 9.47$.
3. $4.83 \cdot \frac{1}{100\%-50\%} = 9.66$.
4. $\frac{padded\_image\_size}{original\_image\_size} = \frac{(176+64\cdot2)\cdot(144+64\cdot2)}{176\cdot144} = 3.26$.
5. 7.5 (fps) / 5.9 (fps) × 140 (MHz) = 178 (Million cycles per second).
6. 5.9 fps/(100%–50%) = 11.8 fps, 50% for MC + Inverse DCT+ loop filtering.

References

1. Joint Video Team (JVT) of ISO/IEC MPEG&ITU-T VCEG, "Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification," JVT-G050, ISO/IEC JTC1/SC29/WG11 and ITU-T SG16 Q.6, Mar. 2003.
2. J. Zhang, Y. He, S. Yangm, and Y. Zhong, "Performance and Complexity Joint Optimization for H.264 Video Coding," in *Proc. IEEE International Symposium on Circuit and System*, May 2003, pp. 888–891.
3. V. Lappalainen, A. Hallapuro and T. D. Hamalainen, "Performance of H.26L Video Encoder on General Purpose Processor," *Journal of VLSI Signal Processing System for Signal, Image and Video Technology*, pp. 239–249, 2003.
4. V. Lappalainen, A. Hallapuro, and T. D. Hamalainen, "Optimized Implementation of Emerging H.26L Video Decoder on Pentium III," in *Proc. 5th WSES*, July 2001.
5. K. Ramkishor and V. Gunashree, "Real Time Implementation of MPEG-4 Video Decoder on ARM7TDMI," in *Proc. IEEE International Symposium on Intelligent Multimedia, Video, and Speech Processing*, May 2001, pp. 522–526.
6. M.-H. Zhou and R. Tallurt, "DSP-Based Real Time Video Decoding," in *Proc. IEEE International Conference on Consumer Electronics*, pp. 296–297, 1999.
7. W. Lin, et al., "Real Time H.263 Video Codec Using Parallel DSP," *Proc. IEEE International Conference on Image Processing*, pp. 586–589, 1997.
8. H. Fujiwara, et al., "An All-ASIC Implementation of a Low Bit-Rate Video Codec," *IEEE Trans. on Circuit and System for Video Technology*, vol. 2, pp. 123–134, June 1992.
9. M. Harrand, et al., "A Single Chip CIF 30-Hz, H.261, H.263, and H.263+ Video Encoder\Decoder with Embedded Display Controller," *IEEE Journal of Solid State Circuits*, vol. 34, pp. 1627–1633, Nov. 1998.
10. M. Ikeda, et al., "SuperEnc: MPEG-2 Video Encoder Chip," *IEEE Micro magazine*, July 1999.
11. M. Berekovic, et al., "Muiticore System-on-Chip Architecture for MPEG-4 Streaming Video," *IEEE Trans. on Circuit and System for Video Technology*, vol. 12, no. 8, pp. 688–699, 2002.
12. S.-K. Jang, et al., "Hardware-Software Co-Implementation of a H.263 Video Codec," *IEEE Trans. on Consumer Electronics*, vol. 46, no. 1, pp. 191–200, 2000.
13. L. D. Soares, S. Adachi, and F. Pereira, "Influence of Encoder Parameter on the Decoded Video Quality for MPEG-4 over W-CDMA Mobile Network," in *Proc. IEEE International Conference on Image Processing*, 2000, pp. 148–151.
14. S.-H. Wang, C.-N. Wang, G.-Y. Lin, T. Chiang, and H.-F. Sun, "AHG Report on Editorial Convergence of MPEG-4 Reference Software," *ISO/IEC JTC1/SC29/WG11, M9073*, Dec. 2002.
15. Telenor Research, "Video Codec Test Model (TMN5)," ITU-T SG 15, WP 15/1, Experts Group on Very Low Bit Rate Visual Technology, Jan 31, 1995. (Internet: ftp://bonde.na.no/pub/tmn/software/tmn-2.0.tar.gz, tmndec–2.0.tar.gz)
16. Joint Video Team (JVT) of ISO/IEC MPEG&ITU-T VCEG, "Context-adaptive VLC (CVLC) Coding of Coefficients," JVT-C028, ISO/IEC JTC1/SC29/WG11 and ITU-T SG16 Q.6, May 2002.
17. S.-H. Wang, et al., "A Platform-Based MPEG-4 Advanced Video Coding Decoder with Block Level Pipelining," in *Proc. IEEE ICICS-PCM*, Singapore, Dec. 2003.

Shih-Hao Wang was born in Tainan, Taiwan, R.O.C. in 1977. He received the M.S. degree in Electrical and Control Engineering from National Chiao Tung University, Hsinchu, Taiwan, in 2001, where he is currently working toward the Ph.D. degree in the Institute of Electronics.

His research interests are video compression and VLSI implementation.

shwang.ee90g@nctu.edu.tw

Wen-Hsiao Peng was born in Hsin-Chu, Taiwan, Republic of China, in 1975. He received the B.S. and the M.S. degrees in Electrics Engineering from National Chiao-Tung University, Hsin-Chu, Taiwan, in 1997 and 1999 respectively. During 2000–2001, he was an intern in Intel Microprocessor Research Lab, U.S.A. In 2002, he joined the Institute of Electronics of National Chiao-Tung University, where he is currently a Ph.D candidate. His major research interests include scalable video coding, video codec optimization and platform based architecture design for video compression applications. Since 2000, he has been working with video coding development and implementation. He has actively contributed to the development of MPEG-4 Fine Granularity Scalability (FGS) and MPEG-21 Scalable Video Coding (Now, MPEG-4 Part 10 AVC Amd.1).

pawn@royals.ee.nctu.edu.tw

Yu-Wen He received his Ph.D. degree in computer application from Tsinghua University in 2002. He was a lecture of the Department of Computer Science and Technology from 2002 to 2003 in Tsinghua University. In 2004, he joined Internet Media group of Microsoft Research Asia.

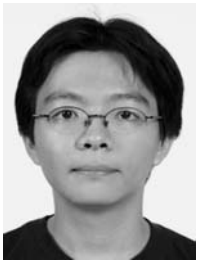
His research interests include video coding, transmission and embedded multimedia application systems.

t-ywhe@microsoft.com

Guan-yi Lin was born in Kaohsiung, Taiwan in 1981. He received the B.S. degree in Electronics Engineering from National Chiao Tung University, Hsinchu, Taiwan, in 2003, where he is currently working toward the M.S. degree in the Institute of Electronics.

His research interests are video compression and communication systems design.

a123.ee88@nctu.edu.tw



Cheng-Yi Lin was born in Tainan, Taiwan in 1981. He received the B.S. degree in Electronics Engineering from National Chiao Tung University, Hsinchu, Taiwan, in 2003, where he is currently working toward the M.S. degree in the Institute of Electronics.

His research interests are on-chip communication and testing.

osban.ee92g@nctu.edu.tw

Shih-Chien Chang was born in Taichung, Taiwan in 1981. He received the B.S. degree in Electronics Engineering from National

Chiao Tung University, Hsinchu, Taiwan, in 2003, where he is currently working toward the M.S. degree in the Institute of Electronics.

His research interests are video compression and VLSI implementation.

shihchien.ee92g@nctu.edu.tw

Chung-Neng Wang was born in PingTung, Taiwan, in 1972. He received the B.S. degree and Ph.D degree in computer science and information engineering from National Chiao-Tung University (NCTU), HsinChu, Taiwan in 1994 and 2003, respectively. He joined the faculty at National Chiao-Tung University in Taiwan, R.O.C in January 2003.

Since 2001 he has actively participated in ISO's Moving Picture Experts Group (MPEG) digital video coding standardization process. He has made more than 18 contributions to the MPEG committee over the past 4 years. He published over 23 technical journal and conference papers in the field of video and signal processing. His current research interests are video/image compression, motion estimation, video transcoding, and streaming.

Tihao Chiang was born in Cha-Yi, Taiwan, Republic of China, 1965. He received the B.S. degree in electrical engineering from the National Taiwan University, Taipei, Taiwan, in 1987, and the M.S. degree in electrical engineering from Columbia University in 1991. He received his Ph.D. degree in electrical engineering from Columbia University in 1995. In 1995, he joined David Sarnoff Research Center as a Member of Technical Staff. Later, he was promoted as a technology leader and a program manager at Sarnoff. While at Sarnoff, he led a team of researchers and developed an optimized MPEG-2 software encoder. For his work in the encoder and MPEG-4 areas, he received two Sarnoff achievement awards and three Sarnoff team awards.

Since 1992 he has actively participated in ISO's Moving Picture Experts Group (MPEG) digital video coding standardization process with particular focus on the scalability/compatibility issue. He is currently the co-editor of the part 7 on the MPEG-4 committee. He has made more than 90 contributions to the MPEG committee over the past 10 years. His main research interests are compatible/scalable video compression, stereoscopic video coding, and motion estimation. In September 1999, he joined the faculty at National Chiao-Tung University in Taiwan, R.O.C. Dr. Chiang is currently a senior member of IEEE and holder of 13 US patents and 30 European and worldwide patents. He was a co-recipient of the 2001 best paper award from the IEEE Transactions on Circuits and Systems for Video Technology. He published over 50 technical journal and conference papers in the field of video and signal processing.

tchiang@mail.nctu.edu.tw