PAPER

Hardware Software Co-design of H.264 Baseline Encoder on Coarse-Grained Dynamically Reconfigurable Computing System-on-Chip

Hung K. NGUYEN[†a)], *Nonmember*, **Peng CAO**[†b)], *Member*, **Xue-Xiang WANG**[†], **Jun YANG**[†], **Longxing SHI**[†], **Min ZHU**[††], **Leibo LIU**[††], *and* **Shaojun WEI**[††], *Nonmembers*

SUMMARY REMUS-II (**RE**configurable **MU**ltimedia **S**ystem 2) is a coarse-grained dynamically reconfigurable computing system for multimedia and communication baseband processing. This paper proposes a real-time H.264 baseline profile encoder on REMUS-II. First, we propose an overall mapping flow for mapping algorithms onto the platform of REMUS-II system and then illustrate it by implementing the H.264 encoder. Second, parallel and pipelining techniques are considered for fully exploiting the abundant computing resources of REMUS-II, thus increasing total computing throughput and solving high computational complexity of H.264 encoder. Besides, some data-reuse schemes are also used to increase data-reuse ratio and therefore reduce the required data bandwidth. Third, we propose a scheduling scheme to manage run-time reconfiguration of the system. The scheduling is also responsible for synchronizing the data communication between tasks and handling conflict between hardware resources. Experimental results prove that the REMUS-MB (REMUS-II version for mobile applications) system can perform a real-time H.264/AVC baseline profile encoder. The encoder can encode CIF@30 fps video sequences with two reference frames and maximum search range of $[-16, 15]$. The implementation, thereby, can be applied to handheld devices targeted at mobile multimedia applications. The platform of REMUS-MB system is designed and synthesized by using TSMC 65 nm low power technology. The die size of REMUS-MB is 13.97 mm^2. REMUS-MB consumes, on average, about 100 mW while working at 166 MHz. To my knowledge, in the literature this is the first implementation of H.264 encoding algorithm on a coarse-grained dynamically reconfigurable computing system.

key words: *reconfigurable computing, reconfigurable multimedia system, REMUS-II, coarse-grained dynamically reconfigurable architecture, H.264/AVC encoder*

1. Introduction

H.264/AVC is the latest video coding standard developed jointly by ITU-T Video Coding Experts Group and ISO/IEC Moving Picture Experts Group. It was designed to replace all the past video standards in almost all kinds of applications. Therefore, it defines several different profiles to meet the various requirements for distinct applications. Compared with previous standards, the coding tools of H.264/AVC when used in an optimized mode allow themselves to achieve up to 50% improvement in bit-rate efficiency at the expense of more than two times the computational complexity for a decoder and more than ten times the hardware complexity for an encoder [1]. The reason is that many new features such as quarter-pixel accurate motion estimation with variable block sizes (VBS-ME) and multiple reference frames, intra prediction, integer transformation based on the discrete cosine transform (DCT), alternative entropy coding modes CAVLC or CABAC, in-loop de-blocking filter, etc. are included.

Implementation of H.264 codec for mobile devices gives designers some challenges such as reducing chip area and power consumption, increasing application performance, shortening time-to-market, and simplifying the updating process. Traditional approaches, e.g. Application Specific Integrated Circuits (ASICs), Digital Signal Processors (DSPs), Application-Specific Instruction Set Processors (ASIPs), and Field Programmable Gate Arrays (FPGAs) have been used for implementing the mobile multimedia systems. However, none of them meet all of the above challenges [23]. Recently, a very promising solution was the reconfigurable computing systems that couple processors with Coarse-Grained Reconfigurable Architectures (CGRA). When implementing an application, these systems map kernel functions of the application onto the CGRAs so they can achieve high performance approximately equivalent to that of ASIC while maintaining a degree of flexibility close to that of DSP processors. By dynamically reconfiguring hardware, CGRAs allow many hardware tasks to be mapped onto the same hardware platform, thus reducing the area and power consumption of the design.

To overcome some limitations of conventional microprocessors and fine-grained reconfigurable devices in the field of multimedia and communication baseband processing, we developed a coarse-grained dynamically reconfigurable computing system called REMUS-II. In this paper, we first propose an overall mapping flow that is possible to apply for mapping an arbitrary algorithm onto the platform of REMUS-II system. Next, we present HW/SW (Hardware/Software) co-design of the H.264/AVC baseline profile encoder, which is aimed at mobile multimedia applications, on the platform of REMUS-MB (REMUS-II version for Mobile applications) in detail. Our implementation supports all tools of baseline profile and can encode CIF@30 fps video sequences with two reference frames

Manuscript received June 5, 2012.
Manuscript revised September 24, 2012.

[†]The authors are with National ASIC system Engineering Research Center, Southeast University, Nanjing, China.
[††]The authors are with Institute of Microelectronics, Tsinghua University, Beijing, China.
a) E-mail: hungnvnu@gmail.com
b) E-mail: caopeng@seu.edu.cn (Corresponding author)

DOI: 10.1587/transinf.E96.D.601

Copyright © 2013 The Institute of Electronics, Information and Communication Engineers

and search range $[-16, 15]$ in real-time when operating at 166 MHz. The encoder can produce the compressed video sequences of which quality and bit-rate approximate those of the JM reference software. The die size of REMUS-MB is 13.97 mm^2. REMUS-MB consumes, on average, about 100 mW while working at 166 MHz. The experimental results showed that the implementation meets requirements for high performance, flexibility, and energy efficiency.

The rest of the paper is organized as follows. Related works and problem definition are presented in Sect. 2. In Sect. 3, overview of the REMUS-MB architecture is introduced. The overall mapping flow for mapping an arbitrary algorithm onto the platform of REMUS-II system is proposed in Sect. 3. The Sects. 4 and 5 give implementation of the H.264 encoder on REMUS-MB system in detail. The experimental results and conclusion are given in Sects. 6 and 7, respectively.

2. Related Works and Problem Definition

In the literature, there are a lot of works related to the realization of modules of an H.264/AVC encoder, but there are only several works that proposed a complete design of H.264/AVC encoder. Most of them use ASIC approach [e.g. 2-6], whereas the others use DSP [e.g. 7] or ASIP [e.g. 9, 10] approach.

All of ASIC-based implementations aim at supporting high-resolution video sequences (e.g. 720 pHD or 1080 pHD) due to high-performance advantage of ASIC method. Although the ASIC-based approach can achieve optimization on power consumption, area, and performance, it lacks flexibility. Therefore, ASIC-based designs usually have a higher NRE (Non-Recurring Engineering) cost and longer time-to-market when the system needs upgrading or changing. By contrast, multimedia systems are often designed not only for a specific application but also for multiple applications. This sharing of resources by several applications makes the system cheaper and more versatile. Besides high performance, low power consumption and cost, it also needs flexibility to simplify upgrades without replacing the system. Moreover, mobile multimedia applications often have the limited display resolution and low bit-rate, and just need supporting baseline profile with video formats Common Intermediate Format (CIF = 352×288) and Quarter CIF (QCIF = 176×144). Therefore, for the mobile multimedia applications, DSP, ASIP, and CGRA-based approaches can be used to achieve high flexibility. The work in [7] developed and optimized H.264/AVC video encoder on the TM320DM642 DSP platform. However, because of limitation related to performance of DSP processors, the authors implemented a series of optimization to reduce complexity of the algorithm at the expense of significant degradation in compressed video quality. As an alternative solution, ASIP-based designs were proposed to compromise the advantages of ASIC and DSP approaches. The authors in [9] proposed an ASIP processor for only intra-frame encoding part of H.264/AVC encoder. In [10], the authors proposed an ASIP architecture, which is called VSIP, for implementation of H.264/AVC. To implement a real-time H.264/AVC decoder, VSIP needs coprocessors for high computation-intensive parts such as inter prediction and entropy coding. However, there is not any result related to the encoder to be shown.

Obviously, the ASIP approach also does not offer sufficient computing power for implementation of the H.264 codec, especially the H.264 encoder. To overcome these obstacles, several CGRAs such as RaPiD [11], MorphoSys [12], PACT XPP-III [13], [14], and ADRES [15], [16] have been proposed for modern multimedia and wireless communication applications in the last decade. PACT XPP and ADRES are the most noticeable two architectures. PACT XPP-III [13] consists of an array of three types of coarse-grained Processing Array Elements (PAEs): ALU-PAE, RAM-PAE, and FNC-PAE, and a packet communication network. The ALU-PAEs and RAM-PAEs are combined to produce a dataflow array for processing computation-intensive parts. FNC-PAE is a VLIW-like processor core that is aimed at processing control-oriented parts of the application. ADRES [15] is the reconfigurable system template that tightly couples a VLIW processor and a coarse-grained reconfigurable matrix. The VLIW processor and the coarse-grained reconfigurable matrix are integrated into a single architecture but with two virtual functional views. These CGRAs have been proven to be the promising solution for multimedia processing. Some works have been proposed to map either parts [17], [21] or the whole [14], [16], [19] of H.264 decoding algorithms onto CGRAs. However, to my knowledge, there is not any previous work that is proposed for implementation of the H.264 encoder in the literatures. One of the main reasons is that these CGRAs do not provide enough computing resources for dealing with very high computational complexity of H.264 encoder. For example, to solve high computational complexity of the motion estimation module, the proposed VLSI designs usually adopt a parallel architecture that includes 256 PEs (Processing Elements) and an adder-tree in order to exploit completely inherent parallelism of the algorithm. Whereas, the limited resource amount of the existing CGRA architectures makes them impossible to satisfy real-time processing requirement.

3. Architecture of REMUS-II System

3.1 Overview of the REMUS-II Architecture

REMUS-II, which stands for **RE**configurable **MU**ltimedia **S**ystem 2, is a coarse-grained reconfigurable computing system for multimedia and communication baseband processing. We have developed two versions for REMUS-II: REMUS-HD for High-Definition applications and REMUS-MB for Mobile applications. The main difference between REMUS-HD and REMUS-MB is: REMUS-HD has two RPUs (Reconfigurable Processing Unit), whereas REMUS-MB has only one RPU. The overall architecture of the

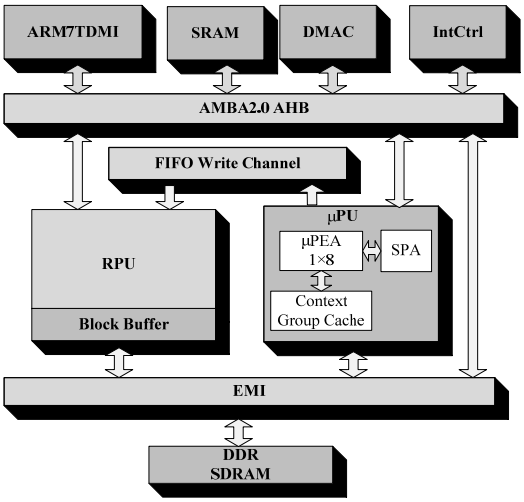


Fig. 1 The overall architecture of REMUS-MB.

REMUS-MB is shown in Fig. 1. The REMUS-MB consists of an ARM7TDMI, an RPU, a Micro Processor Unit (μPU), and several assistant functional modules such as interrupt controller (IntCtrl), Direct Memory Access (DMA) unit, and External Memory Interface (EMI), etc. All modules are connected with each other by ARM AMBA bus. The ARM processor functions as the host processor that is used for controlling application and scheduling tasks. The RPU is a powerful dynamically reconfigurable system, and consists of four Reconfigurable Computing Arrays (RCAs). In turn each of RCAs is an array of 8x8 RCs (Reconfigurable Cells), and can run independently to accelerate computing performance.

In comparison with the previous version [18] of REMUS, this version has been improved by adding the μPU and optimizing Data-flow. The μPU consists of an array of 8 RISC micro-processors (μPEA) and an array of stream processing elements (SPA). The processors of the μPU communicate with each other and with the ARM processor by a simple mailbox mechanism. The processing elements of SPA are dedicated to bit-level computation-intensive applications, e.g. entropy codec, whereas the μPEA is aimed at executing control-intensive parts and float-point operations of applications. With the extended ability of the μPU, HW/SW partition becomes more flexible. By mapping computation-intensive kernel loops onto the RPU, and control-intensive tasks or bit-level, float-point operations onto μPU, the REMUS-II system can achieve the same performance as ASIC while maintaining a degree of flexibility close to that of DSP processors. On the other hand, to satisfy the high data bandwidth requirement of multimedia applications, data flow of REMUS-II was optimized [20] to offer a three-level hierarchical memory, including off-chip memory (DDR SDRAM), on-chip memory (SRAM), and RPU internal memory (RIM). In order to accelerate the data flow, a Block Buffer is also designed to cache data for RPU.

The operation of REMUS-II is reconfigured dynamically in run-time according to the required hardware tasks.

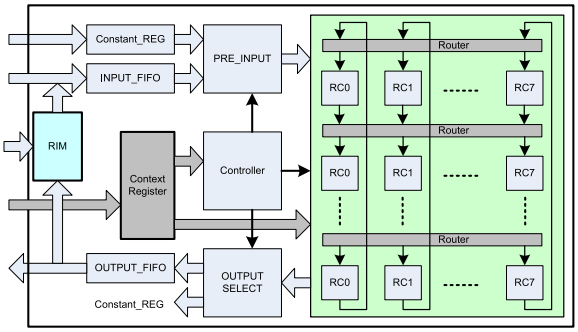


Fig. 2 The RCA architecture.

To support such configuration, RPU is equipped with a Configuration Interface (CI) and a hierarchical configuration memory (CM). The CI is responsible for receiving and buffering configuration words, or context, which are sent from the μPU. Function of whole RCA can be reconfigured once by the CI. The CM is composed of off-chip CM, on-chip CM, and Context Register files. Contexts can be dynamically pre-fetched from off-chip CM and cached by on-chip CM that leads configuration to be accelerated. The Context Register files are equipped for each RCA core in order specify the way each RCA will operate.

3.2 Architecture of RCA Core

The RCA core is composed of an array of 8x8 RCs, an Input FIFO (Input_FIFO), an Output FIFO (Output_FIFO), two Constant Registers (Constant_REG), RIM memory and a controller, etc. (Fig. 2). The input and output FIFO is the I/O buffers between external data flow and RCA. Each RC can get data from the input FIFO or/and Constant_REG, and store data back to the output FIFO. These FIFOs are all 256-bit in width and 8-row in depth, and can load/store thirty-two bytes or sixteen 16-bit words per clock cycle. Especially, the input FIFO can broadcast data to every RC that have been configured to receive the data from the input FIFO. This benefits data-reuse between iterations. Interconnection between two neighboring rows of RCs is implemented by the router. Through the router, an RC can get results that come from an arbitrary RC in the immediately above row of it. The Controller generates the control signals that maintain execution of RCA accurately and automatically according to configuration information in the Context Registers. The architecture of RCA core is basically loop-oriented one. Executing model of the RCA core is pipelined multi-instruction-multi-data (MIMD) model. In this model, each RC can be configured separately to process its own instructions, and each row of RCs corresponds to a stage of pipeline. Multiple iterations of a loop are possible to execute simultaneously in the pipeline.

RC is the basic processing unit of RCA. Each RC includes the data-path that can execute signed/unsigned fixed-point 8/16-bit operations with two/three source operands, such as arithmetic and logical operations, multiplier, and multimedia application-specific operations (e.g. barrel shift,

shift and round, absolute differences, etc.). Each RC also includes a register called TEMP_REG. This register can be used either to adjust operating cycles of the pipeline when a loop is mapped onto the RCA, or to store coefficients during executing an RCA core loop.

3.3 Three-Level Configuration Model

The configuration information for REMUS-II is organized in three levels: RPU-level context (CL0), RCA-level context (CL1), and Core-level context (CL2). The CL0 context consists of the information about which CL1 context will be loaded into the RPU, as well as how to communicate data between internal and external components of the RPU, etc. The total length of CL0 is variable from 1 to 33 32-bit words. The CL0 is dynamically generated by the host ARM or the μPE which is specified as the supervisor of the RPU. The CL1 context defines the data communication of a certain RCA, and the address of CL2 contexts that needs to be loaded for an execution session. The total length of CL1 is variable from 5 to 73 32-bit words. The CL2 context specifies particular operation of the RCA core (operation of each RC, interconnection between RCs, input source, output location, etc.) as well as the control parameters that control operation of the RCA core. The total length of CL2 is 107 32-bit words. The configuration process is divided into three stages: the first stage is for generating configuration context (CL0); the second stage is for pre-fetching and delivering context group (CL1); and the third stage is for reallocating core contexts (CL2). These three stages also can be pipelined.

4. Mapping Methodology

To utilize the abundant amount of resources of REMUS-II for exploiting inherent multilevel parallelism in applications, a mapping flow started from a high-level description of an algorithm keeps an important role. Mapping flow for REMUS-II must deal with many aspects of parallel computing and all its associated compiling techniques such as computation and data partition, inter-process synchronization, and code generation, etc.

In this paper, we propose an overall mapping flow, which is based on traditional SoC (System-on-Chip) design flow, for mapping an arbitrary algorithm onto the platform of REMUS-II system as shown in Fig. 3. The presence of reconfigurable resources in the REMUS-II system leads to the need for modifying and extending the conventional design flow so that it focuses on the higher abstraction levels at where most of the important design decisions are given. Since structure of the reconfigurable hardware has been defined, the main characteristics of the RPU have to be taken into account during HW/SW co-design to identify the parts of the applications that are candidates for mapping on the reconfigurable hardware. Afterwards, the mapping flow synthesizes those parts to generate configuration information (or context) instead of the *architecture design* phase as in

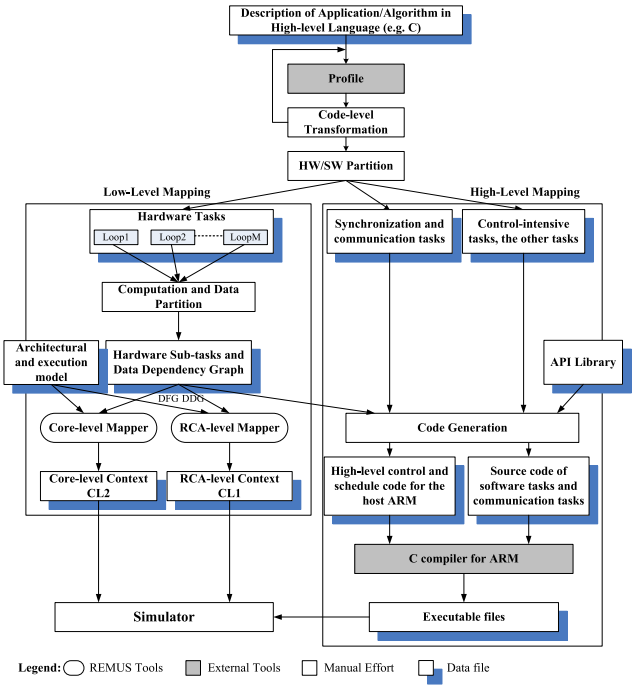


Fig. 3 Mapping flow proposed for REMUS-II system.

the traditional SoC design flow. At this aspect, the mapping flow is quite similar to the compilation flow for software programs running on a processor. On the other hand, run-time reconfigurability of reconfigurable hardware also brings new problems to the partition and mapping of hardware tasks. For partition between hardware tasks, instead of just considering spatial partition as it happens in traditional SoC hardware design the temporal partition and scheduling problem must be addressed. Furthermore, it also needs a mechanism, which is similar to scheduling in multi-task operating systems, to handle context multiplexing as well as inter-context data communication. This causes increase in complexity of the mapping flow, since only mapping of hardware tasks also includes the problems of HW/SW co-design.

We have been working in an effort to automate this mapping flow. Currently, some phases have been assisted by automatic tools (e.g. Core-level Mapper, RCA-level Mapper), whereas the others either have been developing (e.g. partitioning of a large DFG into several sub-DFGs) or still need manual effort (*Code-level transformation*, *HW/SW Partition*, and *Code Generation*).

5. Implementation of H.264 Encoder

The main challenge of H.264 encoder implementation is high computational complexity and huge data bandwidth of some encoding tools, e.g. multi-frame, variable block size motion estimation (MF-VBS-IME). This section presents the method for mapping the H.264 baseline profile encoding algorithm, which is aimed at mobile multimedia applications, onto REMUS-MB system in detail. The REMUS-

II system still lacks the automatic tools that aid designers in the process of *HW/SW Partition* and *Code Generation*. Therefore, to map H.264 encoding algorithm onto the REMUS-MB system, we have done these phases manually using the heuristic method that requires experience and knowledge about the target architecture as well as the algorithm of application. We first develop a REMUS-MB-oriented C-software model for the encoding algorithm and then focus on analyzing computation-intensive loops of the model. Some loop and data structure transformations are used to parallelize computation-intensive parts while reducing inter-process data communication. Next, the computation-intensive loops are mapped onto RPU to increase total computing throughput and solve high computational complexity. Besides, some data-reuse schemes are also used to increase the data-reuse ratio, hence reduce required data traffic. Finally, we propose a scheduling scheme to manage the dynamic reconfiguration of the system. The scheduling scheme also takes charge of synchronizing the data communication between tasks, and managing the conflict between hardware resources.

5.1 Structure of H.264/AVC Encoder for REMUS-MB

The mobile multimedia applications usually have the limited display resolution and the low bit-rate. Therefore, a baseline profile encoder supporting video formats such as CIF and QCIF is appropriate for these applications. However, to achieve the best quality, our encoder supports full-search VBS-ME with two reference frames and a $[-16, 15]$ search range. Some main specifications for the encoder are listed in Table 1. We choose the JM reference software [26] as a start point for developing our C model. The JM software has excellent compression performance, but the code contains a lot of redundant coding tools and the complex data structure. It also includes many sequential processes, so it requires some modifications and optimizations to be mapped efficiently to parallel hardware architecture of the REMUS-MB system.

Table 1 Parameters of H.264 encoder.

Profile	Baseline
Slices	I and P
QP range	20-45
Hadamard Transform	On
RDO Optimization	On
Motion Estimation algorithm	Full Search
Number of Reference Frames	1-2
Motion Vector Resolution	1/4 pixel
Block sizes supported by ME	4×4, 4×8, 8×4, 8×8, 8×16, 16×8,16×16
Search Range	[−16, 15]
Entropy coding	CAVLC, UVLC
Maximum resolution (pixels)	352×288 (CIF)
Frame processing rate	30 frames per second
Bit-rate control	Supported

The C model, which contains only tools of the baseline profile, is profiled by ARM profiler to detect the computational-intensive loops. Figure 4 shows the encoding process of the encoder with main functional blocks: *Motion Estimation* (ME), *Motion Compensation* (MC), *Intra Prediction* (IPRED), *Rate-Distortion Mode Decision* (MD), *(Inverse) Transform and Quantization*, *Reconstruction*, and *Context Adaptive Variable Length Coding* (CAVLC). These functional blocks process 16 × 16-pixel Macro-Blocks (MBs) of each current frame one-by-one. The prediction MB, which is predicted either by Intra Prediction (I-type) or by Inter Prediction (P-type), is subtracted from the current MB to calculate residues. Next, the residues are transformed, quantized, and entropy coded. In our implementation, the computing of residues for I-type MB is merged into MC unit. After that, if MB is P-type, MC unit will continue to interpolate chroma pixels at sub-pixel location according to the motion vector outputted by ME unit and then calculate the chroma residues.

5.2 Hardware/Software Partition

Some time-consuming functional modules as shown in Fig. 4 need accelerating by the RPU, whereas the others are mapped on μPU for processing in parallel.

The ME, which consumes about 50%–90% of the total encoding time depending on the number of reference frames [2], is the most complex module of the encoder. Therefore, the ME is the best candidate for mapping on the RPU. Besides, we also choose MC, Transform and Quantization (*Trans*: including both forward and inverse paths), and Deblocking filter (DB) to map on the RPU due to its inherent instruction-level and data-level parallelism. The loops of these modules are transformed and partitioned fur-

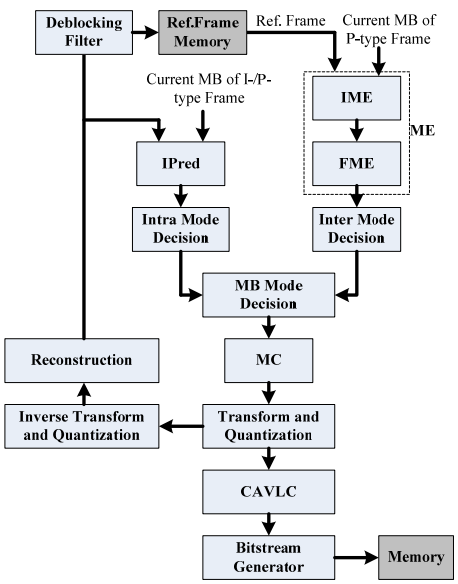


Fig. 4 Functional block diagram of H.264/AVC encoder.

ther into some sub-tasks whose DFG fit on the computing resource of the RPU.

In Intra-frame prediction (IPred), all prediction pixels are calculated based on the reconstructed pixels of previously encoded neighboring blocks or MBs. There are nine prediction modes for luma 4×4-blocks (IPred4×4) and four prediction modes for luma 16×16-block (IPred16×16) and chroma 8×8-blocks (IPred_chroma). For each block, Intra prediction is implemented by referring to its upper, upper-left, left, and upper-right neighboring blocks. Because of data dependency among IPred4×4 of luma 4×4-blocks, five functional units - including *IPRED, IntraMD, TQ, ITQ,* and *ResCons* - form an encoding loop (Fig. 5). Because of supporting various prediction modes without loop-level parallelism, Intra-frame prediction is a control-intensive and irregular process. Therefore, generating prediction pixels for each mode (*IPred16×16*, *IPred_chroma*, and *IPred4×4*), *IntraMD* and *ResCons* are assigned to the μPEA, whereas *TQ* and *ITQ* (i.e. *Trans. for IPred4×4* in Fig. 6) of the IPred4×4 encoding loop are assigned to the RPU.

The parameters of the prediction model (e.g. MB-type, prediction information, QP, etc.), residual coefficients are compressed by the entropy encoder. The *bitstream generator* then produces a compressed bit-stream or file according to syntax of H.264 [1]. A compressed video sequence consists of coded prediction parameters, coded residual coefficients and header information. In the baseline profile of H.264/AVC, CAVLC is used to encode the residual coefficients and Universal Variable Length Coding (UVLC) is used to encode the parameters. All of these methods require bit-level manipulations, which are inefficient to be implemented by the RPU since the RPU is designed to process data at the word-level. For this reason, entropy encoder (*CAVCL and UVLC*) and *bitstream generator* are mapped to the SPA of the μPU.

Finally, control and schedule of the whole encoding process are mapped onto the ARM host processor, whereas the rest of code, including *MVcost generation*, *Rate-Distortion Mode Decision*, *Reconstruction (ReCon)*, *boundary strength (Bs) computation,* etc. are mapped onto the μPEA.

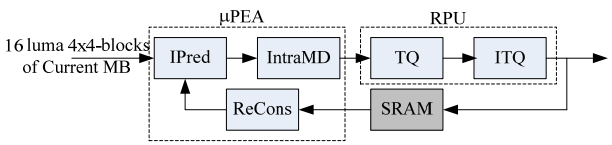


Fig. 5 Encoding loop of IPred4×4.

5.3 Scheduling of Tasks

After HW/SW partition, the hardware tasks are synthesized to generate configuration information (e.g. DFG, control parameters, modes of data transfer). On the other hand, some codes are inserted into the source code of software tasks for purpose of supervision and communication. Each functional module is implemented independently on REMUS-MB system to evaluate functionality and performance before they are integrated together into a complete encoder. Because of run-time reconfigurability of the REMUS-MB system, the integration is a very complex process that requires a good schedule to manage switch tasks in/out of the reconfigurable hardware. The schedule is also responsible for synchronizing the data communication among tasks and handling the conflict between hardware resources.

Based on analyzing the data dependency between tasks of H.264 encoder, the tasks are scheduled and controlled by the host ARM processor. Figure 6 shows scheduling scheme for the worst case, encoding a P-type Macro-block. In order to achieve H.264 CIF@30 fps real-time encoding at 166 MHz clock, the system must process each MB in about 13973 clock-cycles. Firstly, at the system level, data flow of the encoder is shortened by exploiting parallelism among its tasks. In particular, inter prediction is scheduled to execute in parallel with intra prediction, whereas entropy coding and deblocking filter are also scheduled to execute concurrently. At the module level, each task is again parti-

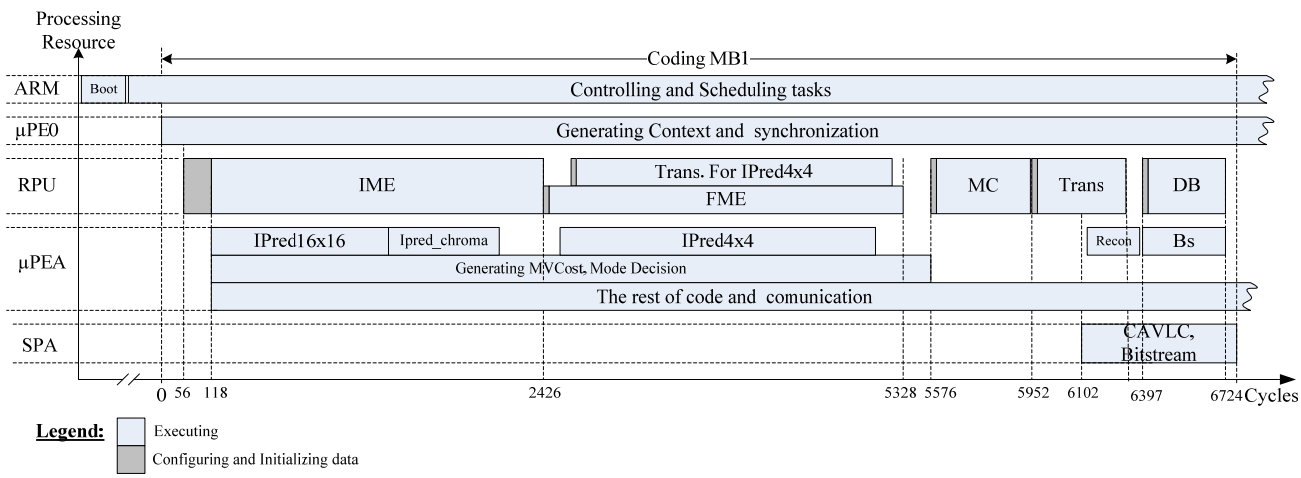


Fig. 6 Scheduling for encoding a P-type Macro-block on the REMUS-MB.

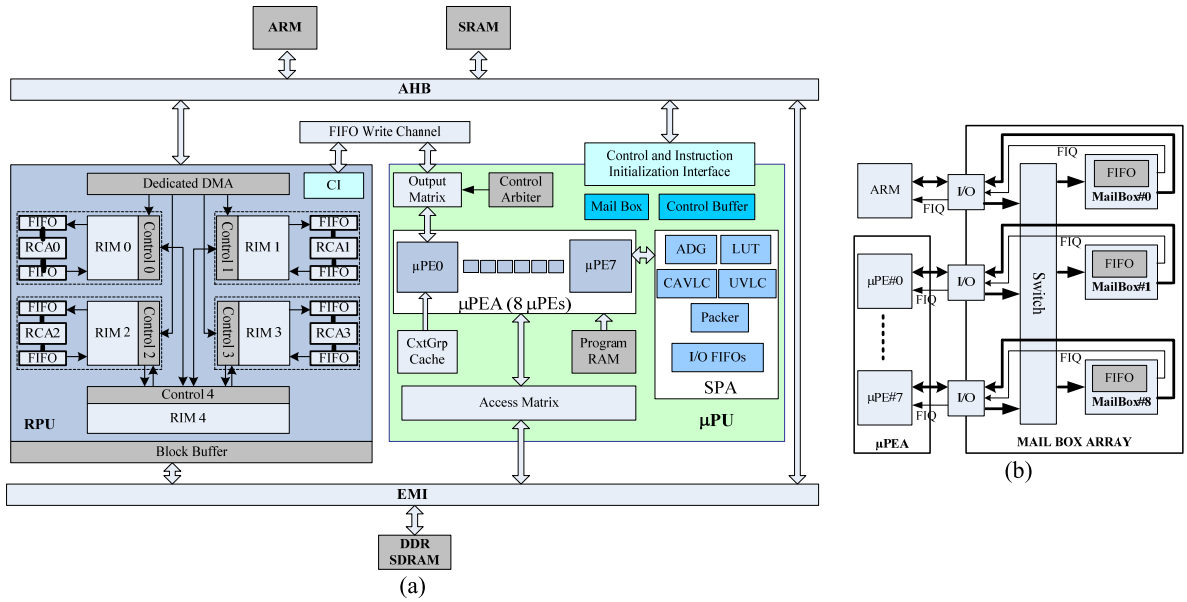


Fig. 7 (a) Configuration of the REMUS-MB for implementation of the whole H.264 encoder, and (b) Mailbox architecture.

tioned into several sub-tasks for parallel execution. For example, SAD/SATD computation of ME task is mapped onto the RPU, whereas generating MVCost, computing the cost function, and making Inter mode decision (InterMD) are assigned to the μPEA; Deblocking process is also divided into two main stages: Boundary-Strength (Bs) parameter calculation which is allocated to the μPEA, and filtering operation that is mapped onto the RPU.

After the system is powered or reset, the ARM host processor is booted automatically from external program memory. It will boot the μPEA then. The main function of ARM is to manage the computing resources of the REMUS-MB by *scheduling* and *allocating* tasks to proper computing resources. Besides, the ARM processor also takes charge of managing the shared memories which are used for data communication among tasks. The μPE of μPEA can operate as a supervisor associated with a RPU or as an independent processing element. As a supervisor, the μPE is responsible for generating or/and selecting configuration words at run-time. As an independent processing element, the μPE is in charge of executing control-intensive tasks and the other tasks which are not suitable for RPUs, e.g. float point operations. In practice, we choose the μPE0 to associate with the RPU as the supervisor that will generate configuration context for the RPU according to the required hardware tasks, as well as implementing data communication between the task on the RPU and the tasks on the other hardware resources. The μPE7 is chosen to supervise and collaborate with the SPA in order to execute the Entropy encoding and generate the output bit-stream (Fig. 7 (a)).

Communication and synchronization: The μPEs of the μPU communicate with each other and with the ARM processor via a simple mailbox-based mechanism (Fig. 7 (b)). When one device (μPE or ARM processor) wants to communicate with the others, it first sends a mail to the mailbox, next the mailbox generates an interrupt that informs the corresponding processor to check the mail and handle it. Since an incoming mail will trigger a new process on μPE, RPU, or SPA, the mail also serves as synchronization.

Operation of RPU: When appearing a requirement to execute a hardware task, e.g. IME task, on the RPU, the ARM processor sends the relevant control information to the μPE0 through the mailbox. Then the μPE0 generates the configuration words dynamically and/or selecting proper configuration context from Context-Group cache to configure the RPU. After receiving configuration context through FIFO Write Channel, communication channel and memory access modes are configured first to pre-fetch data from external memory to RIM and then write to input FIFO. Next, functionalities of RCA cores are configured according to the given task. Once configuration has finished and data is available on inputs (e.g. Input_FIFO, Constant_REGs, TEMP_REGs), RCA core can start execution. Three phases, including Configuration, data Load, and Execution, are possible to implement in pipelined fashion. Therefore, it is possible to hide the time of configuration and data load under the time of execution. The time needed to configure the whole RPU (e.g. for the first time) is about 62 cycles. For the next configuration times (e.g. configuration for *FME* and *Trans*), because the configuration hides under previous processes it only takes four cycles on average to switch between the current task and the next task. Some tasks (for example, *MC* and *DB*) have the data dependence on the processes that are not executed by RPU, thus after configuring they have to wait for available data on inputs. As a result, these tasks cannot start until the previous processes have finished.

Operation of the SPA: After intra/inter prediction and transform and quantization for a MB have completed, MB-

level information (i.e. MB-type, prediction parameters, etc.) and the residual coefficients must be encoded to form the compressed bit-stream. The SPA consists of an array of special-purpose processing elements, which can be used to implement multi-standard entropy encoding/decoding under controlling of one or more μPEs of the μPEA. In this paper, particularly, the μPE7 and the SPA collaborate on encoding the parameters and residual coefficients to form the compressed bit-stream. The μPE7 receives control parameters (e.g. picture parameters, slice-level information, address of residual coefficients, etc.) and generating signals to control operation of the SPA. The SPA is in charge of encoding the residual coefficients by using CAVLC and LUT (Look-up Table) unit, and encoding the parameters by using UVLC unit to produce Header and MB-level bit-streams, respectively. The ADG (ADdress Generating) unit is used to generate addresses that allow residual coefficients to be read from SRAM in the zig-zag order to the input FIFO of the SPA. Besides, SPA is also responsible for packing MB-layer bit-streams and Header information into NAL (Network Abstraction Layer) units of the output bit-stream according to the syntax of the H.264 standard [1]. The whole entropy encoding process requires about 622 cycles to complete for the worst case (QP = 20).

6. Partition and Mapping of Motion Estimation

VBS-ME is the tool that has the highest computational complexity and the largest memory access bandwidth in H.264/AVC encoder. An efficient implementation of ME module has an important role in successful implementation of the whole H.264 encoder. To demonstrate the *low-level mapping* branch of the Mapping flow, this section presents the methodology for mapping the kernel loops of full-search VBS-ME algorithm onto the RPU in detail, as well as showing how the RPU collaborates with the other hardware resources to complete function of VBS-ME algorithm.

Motion Estimation (ME) in video coding exploits temporal redundancy of a video sequence by finding the best matching candidate block of each current MB from a search window in reference frames. The H.264 standard supports VBS-ME that means each current MB has 41 partitions or sub-partitions of seven different sizes (4×4, 4×8, 8×4, 8×8, 8×16, 16×8, and 16×16). The H.264 also supports quarter-pixel accurate ME, so VBS-ME is partitioned into integer ME (IME) and fractional ME (FME). If full-search VBS-IME is chosen and search range is $[-p, p-1]$ in both x- and y-directions, the size of the search window is given by $[N+2p-1, N+2p-1]$. Therefore, each of 41 partitions or sub-partitions needs to be matched with $4p^2$ candidates in the search window by evaluating the RDO (Rate-Distortion Optimized) cost function:

$$J = SAD + \lambda \times MVCost, \tag{1}$$

$$SAD(m,n) = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} |C(i,j) - P(i+m, j+n)|. \tag{2}$$

Here, $SAD(m,n)$ is sum of absolute differences of the current block and the candidate block at search position (m, n), *MVCost* is estimated by using a lookup table defined in the reference software by JVT, and λ is Lagranggian parameter that is derived from the quantization parameter to make trade-off between distortion and bit-rate. $C(x,y)$ and $P(x,y)$ are the current and search pixel, respectively; Motion Vector, MV, is the offset between position of the current macro-block and position of a candidate block in the search window.

Next, FME performs motion search around the center pointed by Integer MV (IMV) and further refines 41 IMVs into fractional MVs (FMVs) of half- or quarter-pixels precision. FME is usually divided into two stages: fractional pixels are interpolated from integer pixels, and then residual pixels between current block pixels and interpolated pixels are used to compute the cost function for each search position. FME interpolates half-pixels using a six-tap filter and then quarter-pixels by a two-tap one [1]. The cost function is given by:

$$J = SATD + \lambda \times MVCost. \tag{3}$$

where, *SATD* is the sum of absolute values of Hadamard transformed residual pixels. The Hadamard transform is based on 4×4-block. Computation of *SATD* is as follows:

$$D(i,j) = C(i,j) - P(i,j) \tag{4}$$

$$\begin{aligned} TD &= H^* D^* H \\ &= \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} {}^* D^* \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \end{aligned} \tag{5}$$

$$SATD = \left(\sum_{i=0}^{3}\sum_{j=0}^{3} |TD(i,j)| \right) \Big/ 2 \tag{6}$$

6.1 Integer Motion Estimation

A. HW/SW Partition

The loops of full-search VBS-IME procedure are transformed so that 41 SADs (in Eq. (2)) of partitions of seven size modes are computed according to a tree-based hierarchical architecture, i.e. SAD of sixteen 4x4-partitions are computed and then reused to compute SADs of larger size partitions. By rearranging, we can reduce about 85% of total amount of SAD computations by eliminating redundant computations. Next, HW/SW partition decides to map the control of the loops onto the ARM7 host processor, while the body of the loops gets further transformation by distributing to two parts: (1) computations of SADs of 41 partitions; and (2) computation of MVcosts and cost function J, and decision on the minimum J. Because computation of MVCost includes bit-level operations, whereas decision on the minimum Js includes many *if-statements*, so it is more efficient to map part (2) onto the μPU. Part (1) is mapped onto the reconfigurable hardware array as shown in the next sub-section.

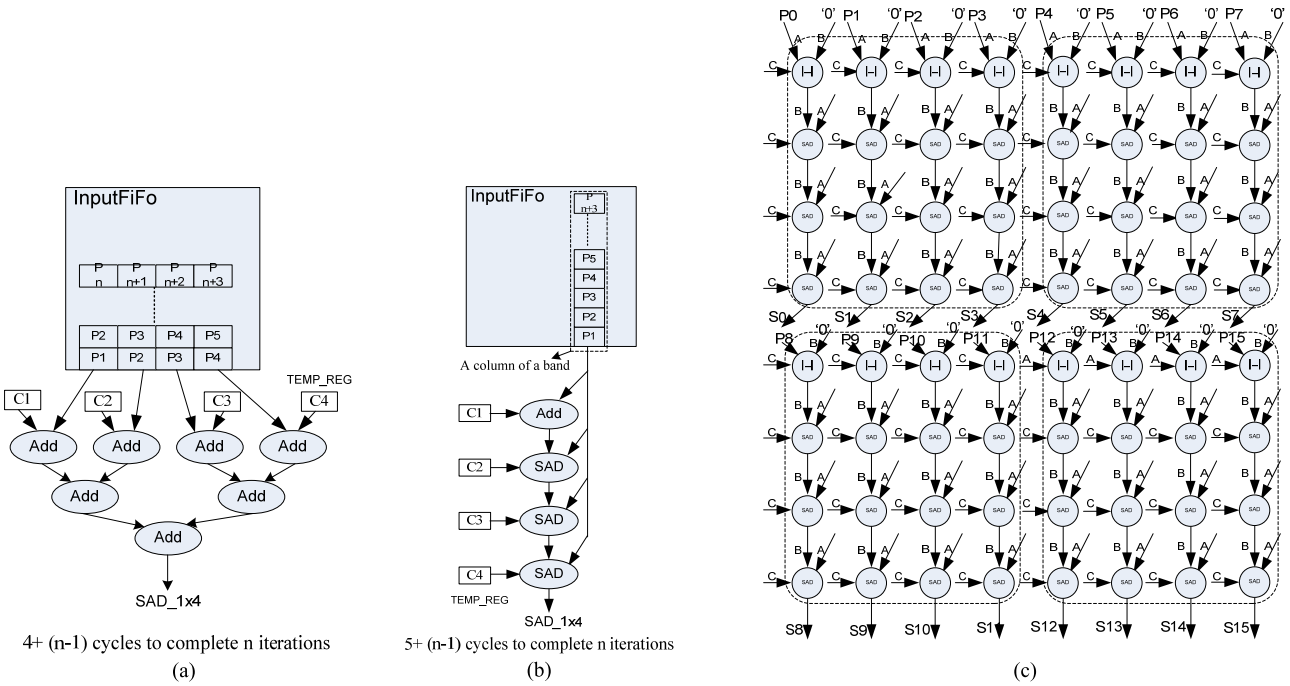


Fig. 8 (a) Parallel implementation, (b) Systolic implementation for computing SAD_1×4 of a candidate, and (c) DFG of the task IME-1 (DFG1) for computing SAD_1×4s of one 8 × 8-block.

B. Partition and Mapping of Hardware Tasks onto RCA

To implement SAD computation of 41 partitions, the proposed VLSI designs usually compute concurrently 256 ADs (Absolute Differences in Eq. (2)) first, and then an adder-tree is used to calculate 41 SADs of partitions/sub-partitions. This process makes a continuous data-flow, so no intermediate data are buffered. In contrast, REMUS-MB just has 256 RCs that are enough to compute 256 ADs concurrently, but for completing computation of 41 SADs it must be implemented by some contexts. In other words, the computation of 41 SADs needs partitioning into several sub-tasks that fit on available resources of the RPU. However, the presence of multi-context gives rise to the problem about inter-context data communication. When partitioning, therefore, we have to consider two criterions: performance and amount of data exchanged between tasks, with constraint of the available (computing and storage) resources of only one RCA. Number of contexts is also taken into account during partition because a large number of contexts increases not only pressure on capacity of memory for intermediate data and configuration contexts but also configuration overhead when switching between contexts.

Firstly, each 16 × 16-MB is divided into four 8 × 8-blocks so that amount of the corresponding computations is fitted in available resource of one 8 × 8-RCA. Then four 8 × 8-blocks are mapped currently onto four 8 × 8-RCAs for parallel processing. Next, the computation of SADs is divided into three tasks: the task IME-1 first computes SAD of 1 × 4-pixel columns (SAD_1×4) at all search positions; next IME-2 is responsible for computing SAD of sub-partitions (i.e. SAD_4×4, SAD_4×8, SAD_8×4, SAD_8×8); and then IME-3 computes SAD of partitions (i.e. SAD_16×8, SAD_8×16, SAD_16×16). Actually, there are some ways to allocate function to IME-1, three ones of them are shown in Table 2. In order to reuse SADs of 4 × 4 sub-blocks for computing larger size blocks, SAD_4×4 is the largest computation granularity that is possible to allocate to IME-1. As shown in Table 2, the way that computes SAD of a 1 × 4-pixel column achieves the best trade-off between the number of operations and amount of data transferred to the next task. Although SAD_1×4 is selected for IME-1, but please note that number of operations is still much larger than available computing resources (only 64 RCs) of the RCA. A full parallel and direct implementation for computation of a SAD_1×4 (as shown in Fig. 8 (a)) is unfeasible due to too resources required. To solve this problem, we propose a systolic-type [22] architecture that utilizes RC's capability of executing three-operand operations to compute SAD_1×4 with only four RCs as shown in Fig. 8 (b). This solution not only allows sixteen SAD_1×4s of a 8 × 8-block to be computed completely by the RCA with only one context but also achieve high performance ($5 + (n - 1)$ cycles for computing n candidates compare with $4 + (n - 1)$ cycles of the solution in Fig. 8 (a)). Moreover, it also exploits fully overlapping data between two successive iterations (i.e. two successive candidates in vertical direction) to reduce inner bandwidth of RPU. Figure 8 (c) shows

Table 2 Various ways of IME-1 task. (for computing an 8×8-candidate)

Granularity of IME-1	AD	SAD_1x4	SAD_4x4
Intermediate results (16-bit)	64	16	4
Operations	64	112	128
Required Contexts	1	1 or 2	2

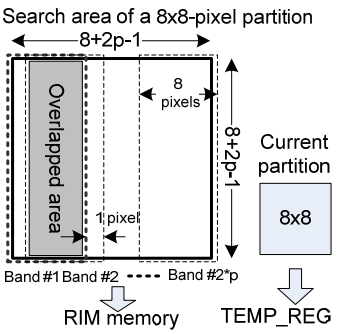


Fig. 9 Input data partition for computing SAD of one 8×8-partition.

DFG of IME-1 (DFG1) for computing one 8×8-block as a demonstrative example. Each DFG1 is in charge of computing concurrently sixteen values of SAD_1×4, and four such DFG1s are possible to be mapped simultaneously onto the RPU for processing whole 16×16-macroblock.

C. Partition and mapping of Data

Partition and mapping of data are required to solve the huge data bandwidth of VBS-IME algorithm by exploiting some data-reuse schemes to increase data-reuse ratio and therefore reduce required data bandwidth. At the beginning of operation, four 8×8-TEMP_REG arrays are loaded 16×16 pixels of current MB and then reconfigured to retain their content during the RCA core is running. Next, pixels of reference area are pre-fetched into the block buffer of the RPU. On the other hand, the distributed RIM modules of the RPU are also exploited to increase the availability of reference pixels for operation of each RCA. Because each RCA is in charge of computing SADs for one 8×8-partition, the reference area as shown in Fig. 9 must be read from external memory and stored in the RIM memory of each RCA. Size of the reference area, $[8 + 2p - 1, 8 + 2p - 1]$ pixels, may be much larger than size of the RIM memory, so the reference area is divided into $2p$ bands of $8 \times (8 + 2p - 1)$ pixels in size, i.e. each band corresponds to one displacement step in horizontal direction and includes all displacement steps in vertical direction. At the particular period of time, only one band is loaded to RIM. When ME process is changed from one band to another band, pixels of the overlapped area can be reused, and only a $(8 + 2p - 1)$-pixel column of the next band must be loaded. By such partition, we can reduce pressure on capacity of the RIM memory as well as the amount of access to external memory. Moreover, in combination with read-while-write functionality of the RIM memory, the time spent for updating the RIM memory is hidden under computing time of the RPU.

6.2 Fractional Motion Estimation

A. HW/SW Partition

In the worst case, the 41 IMVs that are outputted by IME may point to 41 different positions in the reference frame, so each IMV needs to be refined independently into sub-pixel precision. The FME process is implemented sequentially in two steps: half refinement and quarter refinement. There are nine positions searched in both half-pixel refinement and quarter-pixel refinement. In each refinement level, the computing process, in turn, is divided into four parts: sub-pixel interpolation, SATD computation, MVcost generation, and decision on the best candidate. Because computation of MVCost includes bit-level operations, it is more efficient to map this part onto the μPEA. The other parts are mapped onto the reconfigurable hardware as shown in the next sub-section.

B. Partition and Mapping of Hardware Tasks onto RCA

Sub-pixel Interpolation. Because computation of SATD values is based on 4×4-block that is also the smallest size supported by the H.264, therefore, we first focus on mapping computation of a 4×4-block and then reuse it in all types of block size.

Figure 10 (a) shows the DFG of a half-pixel interpolator, which consists of five 6-tap filters. By using TEM_REG registers to buffer data and adjust operating cycles of pipeline, the interpolator can receive ten input pixels and then generate five output pixels simultaneously. The process of half-pixel interpolation first generates horizontal half-pixels and then vertical half-pixels. By using the transpose mode of RIM memory (as shown in Fig. 10 (b)), we can implement interpolation of horizontal pixels as well as vertical pixels with the same DFG of the interpolator. After half-pixels have generated, quarter-pixels are generated by using the DFG shown in Fig. 11. Through the input FIFO, a row of eleven horizontal adjacent integer-pixels and half-pixels are fed into the ten bilinear filters (denoted as BF) in the 1st row for interpolating ten horizontal pixels. The first ten pixels of input FIFO are also shifted down in the TEMP_REG of the RCA. After two cycles, twenty bilinear filters in the 2nd row will generate vertical and diagonal pixels by filtering the corresponding pixels from TEMP_REG and the input FIFO.

In order to apply the above method to the blocks of which sizes are larger than 4×4 pixels, we can decompose them into some 4×4-blocks. However, such decomposing results in a large redundant amount of interpolation due to the overlapped area between adjacent interpolating windows as shown in Fig. 13. To overcome this problem, when filtering horizontal pixels, we decompose a large partition into some blocks of 4-pixel in width and 4/8/16-pixel in height, instead of decomposing it into some 4×4-pixel-fixed blocks. For example, an 8×8-partition is decomposed into two 4×8-blocks, instead of four 4×4-blocks, as shown in Fig. 13 (a). The same technique is also applied for filtering vertical pixels as shown in Fig. 13 (b). Consequently, 24% of cycles required for interpolation is reduced.

SATD Computation. 4×4-Hadamard transform is the most important unit of SATD computation. Many VLSI designs for SATD computation implement 4×4-Hadamard transform unit dependently by using two 1-D Hadamard units (e.g. [25]) without considering correlation between Eq. (4–6). Such implementation is not useful for mapping onto REMUS-MB. In the paper, we have considered the corre-

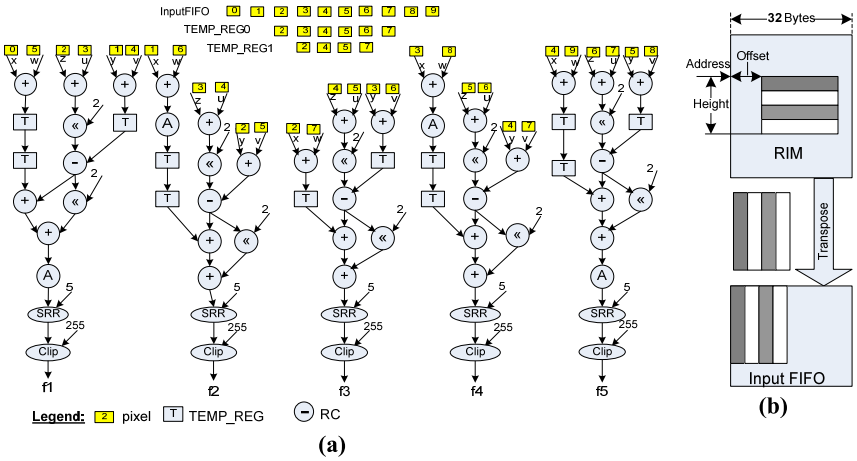


Fig. 10 DFG for 1/2-pel interpolation (a) and transpose mode of RIM (b).

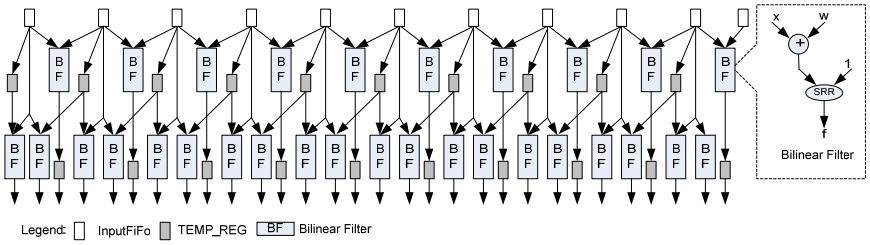


Fig. 11 DFG of Quarter-pixel Interpolator.

lation among Eq. (4–6) in order to find the DFG that is the most optimal for mapping onto the REMUS-MB system. Instead of using two 1-D Hadamard units, we are also using one 2-D Hadamard unit. The optimized DFG for computing SATD, which can be mapped completely onto one 8 × 8-RCA, is shown in Fig. 12. A 4 × 4-pixel block is first divided into two halves of 8 pixels to input to the DFG in sequential order. Eight RCs in the 1st row generate eight residues in parallel and then transmit them to 2-D Hadamard transform unit. The transformed residues of the 1st half are stored in TEMP_REGs waiting for transformed residues of the 2nd half. Once residues of the 2nd half have transformed, they are compared with the transformed residues of the 1st half to find maximum values. The maximum values then are transferred to the adder-tree in order compute SATD value, finishing computing SATD of a 4 × 4-block. The process is fully pipelined with eight latency cycles. No intermediate data is buffered when computing SATD of a 4 × 4-block, therefore, no additional internal memory is required.

6.3 Scheduling of Sub-Tasks

Execution of sub-tasks of ME is scheduled to implement on the hardware resources of REMUS-MB in Time-Division Multiplexing fashion as shown in Fig. 14. Firstly, the first two tasks, IME-1 and IME-2, occupy whole RPU during the first two phases of ME process. Once IME-1 and IME-2 have finished, RPU is reconfigured so that IME-3, Interpolation, and SATD computation are mapped onto RCA0, {RCA1, RCA2}, and RCA3, respectively. Execution and

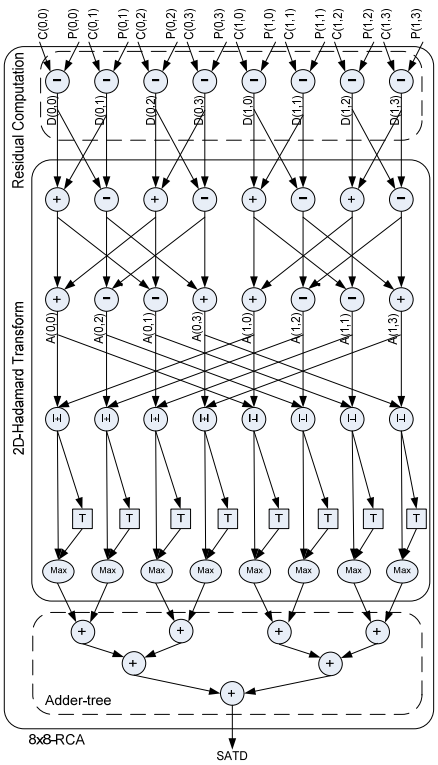


Fig. 12 The DFG for SATD computation.

Data-flow of the RPU are reconfigured dynamically under controlling of μPE0. To reduce computing load on the RPU, the other tasks including *MVCost generation*, *finding the*

best IMV/FMV and *Mode Decision* (*MD*) are assigned to the μPE2 and μPE3 of the μPEA, respectively. The tasks that are mapped onto the RPU communicate data to each other via the distributed RIM memory. On the other hand, on-chip SRAM is used for communicating data between tasks on the μPEA as well as between tasks on the μPEA with tasks on the RPU.

Cooperation among RPU, μPE2 and μPE3 are synchronized by the mailbox mechanism of which operation based on sending and receiving mails. We would describe handshake protocol between IME-2 and μPE3 as a demonstrative example. When IME-2 in progressing and a data block of SAD results has been stored in SRAM, a mail containing address and size of data block will be sent from μPE0 to the μPE3. To prevent new output data from overwriting the old ones, μPE0 also needs to keep the storage area occupied by this data block as "write-protected area" in SRAM. Similarly, the μPE3 also have had address of MVcost values which are generated and mailed to μPE3 by μPE1 before. After receiving mail from μPE0, μPE3 loads the SADs and MVcost values from the corresponding addresses of SRAM and then computes on the data to find the best IMV. When the whole data block in SRAM has been copied into data memory of μPE3, even if it has not been processed by μPE3, μPE3 will sent back a mail to μPE0 and μPE2 in order inform them that the storage area occupied by the data block in SRAM may be unlocked, and that it is ready to receive next data block.

7. Evaluation and Experiment Results

To evaluate the performance of REMUS-MB, a functional RTL model has been designed in Verilog and then synthesized by Synopsys Design Compiler using TSMC 65 nm low power technology. The die size of REMUS-MB is 13.97 mm^2. REMUS-MB consumes about 100 mW on average while working at 166 MHz. The detailed specifications of the platform are shown in Table 3. The layout of REMUS-MB is given in Fig. 15.

To encode a CIF video sequence at frame rate 30 fps (CIF@30 fps) in real-time, the system that operates at 166 MHz has a 13973-cycle budget for encoding a MB. Because of target at mobile multimedia applications, some performance evaluations are implemented on QCIF and CIF video sequences, which contain 120 frames at frame rate 30 fps, by RTL simulation. Performance of the kernel loops, which are mapped on the RPU, is shown in Table 4. The experimental results showed that, in the worst case, the number of cycles to encode a P-type MB by our implementation on the REMUS-MB platform is 6750 cycles with search range $[-16, 15]$ for ME. The REMUS-MB system is completely able to encode CIF@30 fps video sequences with two reference frames and search range $[-16, 15]$ when operating at 166 MHz.

To evaluate the compressed video quality, we also compare our implementation with the reference software JM18. The JM software encoder is established to the same set of encoding parameters as in Table 1. Both our encoder and JM

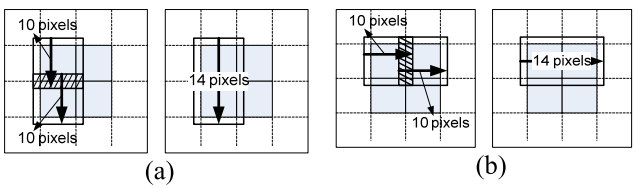


Fig. 13 The overlapped area among adjacent interpolating windows in horizontal (a) and vertical (b) direction.

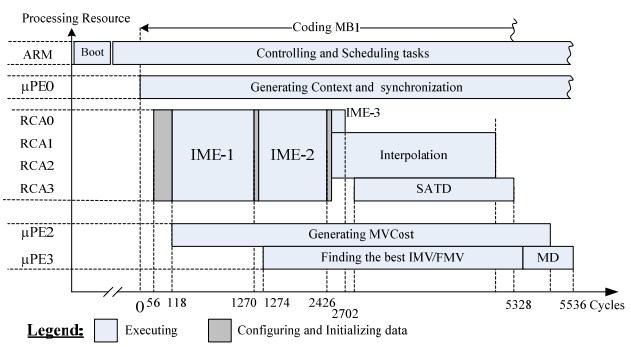


Fig. 14 Scheduling of sub-tasks of ME on REMUS-MB.

Table 3 Specifications of the REMUS-MB platform.

Operating Frequency		166 MHz
Average Power Consumption		100 mW
Core Voltage		1.2 V
Core Area	RPU	13.09 mm^2
	µPEA	0.34 mm^2
	SPA	0.54 mm^2
	total	13.97 mm^2
Technology		TSMC 65nm low power
Logic gates		7,740,117
SRAM		383.53 KB

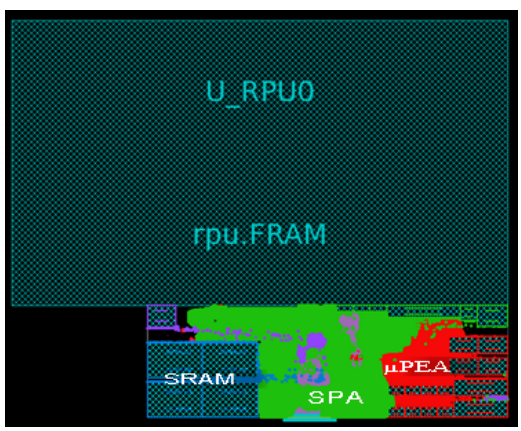


Fig. 15 The layout of REMUS-MB.

reference encoder are tested with three CIF@30 fps video sequences: news, waterfall, and mobile (which are selected according to motion level from low to high). The PSNR versus Bitrate curves are shown in Fig. 16. Our implementation can produce the compressed video sequences with quality and bit-rate which approximate those of the JM18: for the worse case quality degradation is about 0.1 dB.

Table 5 shows performance comparison between our implementation and the experimental results of several previous works. The PSNR values are measured at the same input video sequence ("Foreman" sequence) and parameter QP = 28. Our implementation is the first CGRA-based H.264 encoder reported in the literature. Our implementation can support real-time encoding CIF@30 fps video sequences with all encoding tools of baseline profile such as quarter-pixel resolution VBS-ME with one or two reference frames, intra prediction, DCT and Hadamard transform, RDO mode decision, CAVLC, etc. Therefore, our implementation can achieve the highest compression performance.

Table 4 Performance of H.264 Encoding Kernel Loops on the RPU.

Loop	Cycles/MB
IME-1	1152
IME-2	1152
IME-3	272
Interpolation	2196
SATD	2072
MC	372
Trans.	350
Deblocking	316

8. Conclusion and Future Works

The paper proposes the overall methodology for mapping algorithms onto the platform of REMUS-MB system and then presents the work on the HW/SW co-design of a real-time H.264 baseline profile encoder. At the *low-level mapping* branch of the mapping flow, we present methodology for mapping a highly complex algorithm as full-search VBS-ME algorithm in detail. Because the REMUS-MB system is designed with computing resources for a range of applications, so it takes a lot of effort to partition the encoding process into several tasks/sub-tasks, which fit on available computing resources of the system, and then gives them a proper scheduling scheme to synchronize their co-

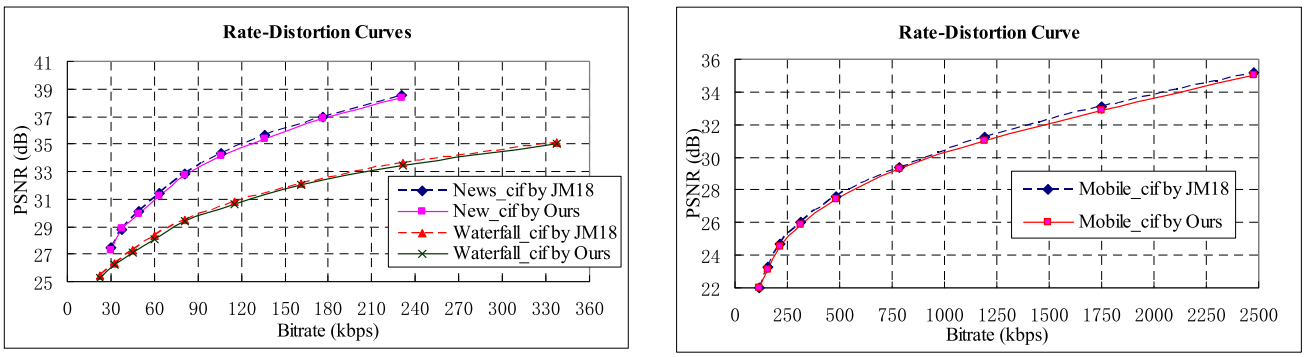


Fig. 16 The Rate-Distortion curves of (a) News_cif and Waterfall_cif, and (b) Mobile_cif video sequences.

Table 5 Performance comparison.

Architecture		[8]	[24]	[7]	[9]	REMUS-MB
Design Approach		Intel PXA27x	TM320DM642 DSP	TM320DM642 DSP	ASIP	CGRA
Profile		Baseline profile	Baseline profile	Baseline profile	-	Baseline profile
Specification		QCIF@25fps	CIF@28fps	CIF@30fps	Only Intra prediction	CIF@30fps
Supported Features		N/A	N/A	N/A		All baseline profile encoding tools
RDO		Off	Off	Off	N/A	On
ME	Algorithm	Fast	Fast	Fast	-	Full search
	Search range	N/A	N/A	N/A	-	[−16,15]
	Ref. frames	1	1	1	-	2
PSNR @QP=28, Foreman		35.9	N/A	33.4	-	37.1
F_{clk} (MHz)		624	600	720	300	166
Average Power (mW)		N/A	N/A	N/A	N/A	100

operation. After mapping, we simulated to validate both functionality and timing of whole implementation. Our mapping methodology exploits data reuse and computation parallelism of the algorithm in order not only to increase total computing throughput and solve high computational complexity but also to reduce the number of configuration switches and inter-process data communication between tasks. Experiments in mapping H.264 encoding algorithm onto REMUS-MB system demonstrate that complex applications can be mapped with competitive performance on REMUS-MB platform. Performance evaluation shows that REMUS-MB can perform H.264 encoder at a real-time speed for CIF@30 fps video sequences with two reference frames and maximum search range of $[-16, 15]$. The implementation, therefore, can apply to handheld devices targeted at mobile multimedia applications such as video recorder, video telephone, video conference, etc. To my knowledge, this is first-time HW/SW co-design of the H.264 encoder on CGRA is reported in the literature.

In this paper, to meet requirements of low power and area for mobile applications, we have used the REMUS-MB to implement the encoder. For HDTV applications, we also have developed the REMUS-HD platform, which includes two RPUs. With the abundant resources of the REMUS-HD, the encoding process can be organized in multi-stage pipelined architecture to process concurrently more than one MB at the same time and therefore improve the hardware utilization and the performance. In the future, some aspects such as task partition, DFG of each sub-task, scheduling scheme, etc., will continue to be optimized according to the architecture of the REMUS-HD to achieve higher performance at higher resolutions. Besides, to extend applicability of the REMUS-II system, we also will focus on developing tools to automate the proposed mapping flow. Moreover, to aid designers in efficiently and conveniently using the REMUS-II system, it is necessary to have an operation system (OS) which handles and synchronizes the data communication among tasks and takes care of hardware resource conflicts.

Acknowledgments

This work was supported by the National High Technology Research and Development Program of China (863 Program) (grant no.2009AA011700) and the National Natural Science Foundation of China (grant no.61204023, grant no.61006029, and grant no.61203251).

The authors would like to thank to C. MEI, B. LIU, JJ. YANG, YC. LU, YQ. FAN, GG. GAO, J. XIAO, H. LEI and CX. ZHANG for their helpful discussions and technical support.

References

[1] I.E. Richardson, The H.264 advanced video compression standard, second edition, John Wiley & Sons, 2010.

[2] T.C. Chen, S.Y. Chien, Y.W. Huang, C.H. Tsai, C.-Y. Chen, T.-W. Chen, and L.-G. Chen, "Analysis and architecture design of an HDTV720p 30 frames/s H.264/AVC encoder," IEEE Trans. Circuits Syst. Video Technol., vol.16, no.6, pp.673–688, June 2006.

[3] H.-C. Chang, J.-W. Chen, C.-L. Su, Y.-C. Yang, Y. Li, C.-H. Chang, Z.-M. Chen, W.-S. Yang, C.-C. Lin, C.-W. Chen, J.-S. Wang, and J.-I. Quo, "A 7 mW-to-183 mW dynamic quality-scalable H.264 video encoder chip," IEEE International Conference on Solid-State Circuits, Digest of Technical Papers, pp.280–281, San Francisco, USA, Feb. 2007.

[4] K. Babionitakis, G. Doumenis, G. Georgakarakos, G. Lentaris, K. Nakos, D. Reisis, I. Sifnaios, and N. Vlassopoulos, "A real-time H.264/AVC VLSI encoder architecture," J. Real-Time Image Processing 2008, vol.3, no.1–2, pp.43–59, DOI: 10.1007/s11554-007-0054-9.

[5] Z. Liu, Y. Song, M. Shao, S. Li, L. Li, S. Ishiwata, M. Nakagawa, S. Goto, and T. Ikenaga, "HDTV1080p H.264/AVC encoder chip design and performance analysis," Proc. IEEE Symposium on VLSI Circuits, pp.12–13, Kyoto, Japan, June 2007.

[6] Y.-K. Lin, D.-W. Li, C.-C. Lin, T.-Y. Kuo, S.-J. Wu, et al., "A 242-mW 10-mm^2 1080p H.264/AVC high-profile encoder chip," IEEE International Solid State Circuits Conference, Digest of technical papers, pp.314–615, San Francisco, USA, Feb. 2008.

[7] D.-T. Lin and C.-Y. Yang, "H.264/AVC Video encoder realization and acceleration on TI DM642 DSP," Lect. Notes Comput. Science, 2009, vol.5414/2009, pp.910–920, DOI: 10.1007/978-3-540-92957-4_79.

[8] Z. Wei, K.L. Tang, and K.N. Ngan, "Implementation of H.264 on Mobile Device," IEEE Trans. Consum. Electron., vol.53, no.3, pp.1109–1116, Aug. 2007. DOI: 10.1109/TCE.2007.4341593.

[9] K. Kim, S. Park, and Y. Paek, "Application-specific instruction set processor for H.264 on-chip encoder," SoC Design Conference (ISOCC), 2009 International, pp.373–376, DOI: 10.1109/SOCDC.2009.5423839.

[10] S.D. Kim and M.H. Sunwoo, "ASIP Approach for Implementation of H.264/AVC," J. Signal Processing Systems, vol.50, no.1, pp.53–67, 2006. DOI: 10.1007/s11265-007-0109-y.

[11] C. Ebeling, D.C. Cronquist, P. Franklin, J. Secosky, and S.G. Berg, Mapping Applications to the RaPiD Configurable Architecture, FPGAs for Custom Computing Machines, IEEE, 1997.

[12] H. Singh, M.H. Lee, G. Lu, F.J. Kurdahi, N. Bagherzadeh, and E.M. Chaves Filho, "MorphoSys: An integrated reconfigurable system for data-parallel and computation-intensive applications," IEEE Trans. Comput., vol.49, no.5, pp.465–481, 2000.

[13] X. Technologies, "XPP-III processor overview," White Paper, July 2006.

[14] M.K.A. Ganesan, S. Singh, F. May, and J. Becker, "H.264 decoder at HD resolution on a coarse grain dynamically reconfigurable architecture," International Conference on Field Programmable Logic and Applications, pp.467–471, 2007.

[15] B. Mei, S. Vernalde, D. Verkest, H. Man, and R. Lauwereins, "ADRES: An architecture with tightly coupled VLIW processor and coarse-grained reconfigurable matrix," Field Programmable Logic and Application, Lect. Notes Comput. Sci., vol.2778, pp.61–70, Springer, 2003.

[16] B. Mei, B. De Sutter, T. Vander Aa, M. Wouters, A. Kanstein, and S. Dupont, "Implementation of a Coarse-Grained reconfigurable media processor for AVC decoder," J. Signal Processing System, vol.51, pp.225–243, 2008.

[17] C. Arbelo, A. Kanstein, S. Lopez, J.F. Lopez, M. Berekovic, R. Sarmiento, and J.-Y. Mignolet, "Mapping control-intensive video kernels onto a coarse-grain reconfigurable architecture: The H.264/AVC deblocking filter," in Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE '07, pp.1–6, 2007.

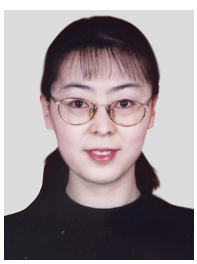
[18] M. Zhu, L. Liu, S. Yin, C. Yin, and S. Wei, "A cycle-accurate simulator for a reconfigurable multi-media system," IEICE Trans. Inf. & Syst., vol.E93-D, no.12, pp.3202–3210, Dec. 2010.

[19] M. Zhu, et al., "A reconfigurable multi-processor SoC for media applications," IEEE International Symposium on Circuits and Systems,

2010 (ISCAS 2010).
[20] X. Liu, C. Mei, P. Cao, M. Zhu, and L. Shi, "Date flow optimization of dynamically coarse grain reconfigurable architecture for multimedia applications," IEICE Trans. Inf. & Syst., vol.E95-D, no.2, pp.374–382, Feb. 2012.
[21] X. Yang, L. Liu, S. Yin, M. Zhu, W. Jia, and S. Wei, "Mapping deblocking algorithm of H.264 decoder onto a reconfigurable array architecture," 2011 International Conference on Consumer Electronics, Communications and Networks (CECNet), pp.4166–4169, 2011.
[22] H.T. Kung and P.L. Lehman, "Systolic (VLSI) arrays for relational database operations," Proc. ACM-Sigmod 1980 Int'l Conf. Management of Data, p.105, 1980.
[23] G. Theodoridis, D. Soudris, and S. Vassiliadis, "A survey of coarse-grain reconfigurable architectures and cad tools basic definitions, critical design issues and existing coarse-grain reconfigurable systems," pp.89–149, 2008.
[24] L. Wang and C.J. Ping, "H.264 video encoder implementation based on TMS320DM642 DSP," International Conference on Management and Service Science, 2009 (MASS '09), pp.1–4, Sept. 2009. DOI: 10.1109/ICMSS.2009.5304159.
[25] T.-C. Chen, Y.-W. Huang, and L.-G. Chen, "Fully utilized and reusable architecture for fractional motion estimation of H.264/AVC," Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, pp.9–12, Montreal, Canada, May 2004.
[26] JM reference software, http://iphome.hhi.de/suering/tml/

Hung K. Nguyen received the B.S. and M.S. degrees in electronic engineering from Vietnam National University, Hanoi, Vietnam, in 2003 and 2005, respectively. He is currently working to get the Ph.D. degree in electronic engineering at Southeast University, Nanjing, China. His research interests mainly include multimedia processing, reconfigurable computing and SoC designs.

Peng Cao received the B.S., M.S. and Ph.D degrees in Information Engineering and Electrical Engineering from Southeast University in 2002, 2005 and 2010 respectively. His research interests mainly include digital signal and image processing, image/video compression, reconfigurable computing and related VLSI designs.



Xue-Xiang Wang was born in Xinjiang Province, China, in 1972. She received the M.S. degree in electronics in 2001 from the Southeast University of China. She is currently pursuing the Ph.D. degree with the Southeast University. Her research interests include SoC design, memory subsystem.

Jun Yang received the B.S., M.S., and Ph.D. degrees from Southeast University, Nanjing, China, in 1999, 2001, and 2004, respectively, all in electronic engineering. He is currently a research fellow and the chairman of SoC department of National ASIC system Engineering Research Center (CNASIC), Southeast University. His research interests include chip architecture design and VLSI design.

Longxing Shi received the B.S., M.S., and Ph.D. degrees from Southeast University, Nanjing, China, in 1984, 1987, and 1992, respectively, all in electronic engineering. He is currently a Professor and the Dean of Integrated Circuit (IC) College, Southeast University. His research interests include system-on-a-chip design, VLSI design, and power IC design.

Min Zhu received the B.S. degree from the Department of Micro & Nano Electronic, Tsinghua University, Beijing, China, in 2006, where he is currently working toward the Ph.D. degree in the Institute of Microelectronics. His research interests include reconfigurable computing and multimedia processing.

Leibo Liu received the B.S. degree in electronic engineering from Tsinghua University, Beijing, China, in 1999 and the Ph.D. degree in Institute of Microelectronics, Tsinghua University, in 2004. He now serves as an Associate Professor in Institute of Microelectronics, Tsinghua University. His research interests include Reconfigurable Computing, Mobile Computing and VLSI DSP.

Shaojun Wei was born in Beijing, China in 1958. He received Ph.D. degree from Faculte Polytechnique de Mons, Belguim, in 1991. He became a professor in Institute of Microelectronics of Tsinghua University in 1995. He is a senior member of Chinese Institute of Electronics (CIE). His main research interests include VLSI SoC design, EDA methodology, and communication ASIC design.