A Cycle-Accurate Simulator for a Reconfigurable Multi-Media System

Min ZHU^{†a)}, Leibo LIU[†], Nonmembers, Shouyi YIN[†], Member, Chongyong YIN[†],
and Shaojun WEI[†], Nonmembers

SUMMARY This paper introduces a cycle-accurate Simulator for a dynamically REconfigurable MUlti-media System, called SimREMUS. SimREMUS can either be used at transaction-level, which allows the modeling and simulation of higher-level hardware and embedded software, or at register transfer level, if the dynamic system behavior is desired to be observed at signal level. Trade-offs among a set of criteria that are frequently used to characterize the design of a reconfigurable computing system, such as granularity, programmability, configurability as well as architecture of processing elements and route modules etc., can be quickly evaluated. Moreover, a complete tool chain for SimREMUS, including compiler and debugger, is developed. SimREMUS could simulate 270 k cycles per second for million gates SoC (System-on-a-Chip) and produced one H.264 1080p frame in 15 minutes, which might cost days on VCS (platform: CPU: E5200@ 2.5 Ghz, RAM: 2.0 GB). Simulation showed that 1080p@30 fps of H.264 High Profile@ Level 4 can be achieved when exploiting a 200 MHz working frequency on the VLSI architecture of REMUS.

key words: *SimREMUS, simulator, reconfigurable multimedia system, compiler, debugger*

1. Introduction

Nowadays the dynamic reconfigurable system is a reality [1], which is superior to GPP (General Purpose Processor) in terms of energy efficiency, and also more flexible than ASIC [2], particularly suitable for running regular and computing-intensive tasks, such as media processing applications, communication baseband applications, etc.

Modern reconfigurable computing systems involve a large number of design options, therefore very complicated tradeoffs need to be taken into account, such as configuration memories and routes. Less configuration memory is necessary since not actually used modules in a system do not allocate configuration memory. Thus it is possible to reduce the necessary configuration area and develop lower-cost and more power efficient systems [1]. Routes mean tradeoff between connect flexibility and area consumption. In order to evaluate the positive aspects of reconfigurable multi-media system, and to decrease the negative impacts of the known tradeoffs, it is necessary to know them as soon as possible in the design flow, which is normally unclear before the system has been programmed on chip during the testing phase. So simulation is important. But lots of processing elements

on reconfigurable multi-media system make system simulation more and more complicated and time-consuming [3]. Data sharing and distributing are always tough problems on parallel computing.

PREProS [1] presented a reconfigurable simulator, and derived XPP [4] architecture. However, power information could not be profiled, and no compiler and debugger techniques were reported. ReSim [3] is a parameterizable ILP (Instruction-Level-Parallelism) processor simulation acceleration engine based on reconfigurable hardware. The drawback of ReSim is that it is not a cycle-accurate simulator therefore detailed performance and behavior could not be profiled and evaluated. Due to the complex configuration, calculation, data transmission operations in reconfigurable computing system, cycle-accurate simulator is highly preferred in the architecture design.

This paper proposes a cycle-accurate simulator for a reconfigurable multi-media system, called SimREMUS, which supports high-level representations to model and simulate the reconfigurable processor. System integration leverages ARM ESL platform [5]. Modules are written in SystemC and cycle-accurate mechanism on each connection of modules is introduced. Many important issues such as granularity, programmability, configurability as well as architecture of processing elements and route modules etc. can be easily evaluated by SimREMUS. The object-oriented concepts as inheritance and polymorphism are also adopted to make simulator easy to maintain and update. Corresponding compiler [6] and debugger are developed, which make up of a complete tool chain for SimREMUS. Run-time dynamic configuration contexts could be generated from the media processing applications programmed in C-language and mapped onto PEA (Processing Element Array) automatically. Therefore, not only reconfigurable SoC architecture could be explored conveniently and evaluated fast, but also media applications could be mapped easily and debugged handily. More concepts can be evaluated by SimREMUS than PREProS simulator such as programmability and route, etc. It provides fast simulation speed like ReSim, but more accurate performance and power information. SimREMUS simulated 270 kcycles per second for this million gates system and produced one 1080p frame in 15 minutes, which might cost days on VCS (Platform: CPU: E5200@2.5 Ghz, RAM: 2.0 GB). Simulation showed that 1080p@30 fps of H.264 HiP@ Level 4 can be achieved when exploiting a 200 MHz working frequency on final architecture.

Manuscript received February 3, 2010.

Manuscript revised June 8, 2010.

[†]The authors are with Institute of Microelectronics, Tsinghua University, Beijing, China.

a) E-mail: zhu-m06@mails.tsinghua.edu.cn

DOI: 10.1587/transinf.E93.D.3202

Fig. 1 The REMUS architecture.

This paper is organized into different sections. Section 2 presents the architecture of a reconfigurable multi-media system, and Sect. 3 introduces SimREMUS simulator including cycle-accurate mechanism, process element modeling and system integration. Compiler and debugger are presented in Sect. 4 and 5 respectively. Section 6 gives out an impression case designed through this approach and Sect. 7 presents the final considerations.

2. Hardware Architecture

REMUS architecture is chosen here due to the characteristics of high-performance media processing applications. High-performance media processing algorithms have regular styles of calculation, detachable controlling and calculation stream etc., therefore a heterogeneous array architecture containing multi-function ALUs, multipliers and conditional branching units is desired. REMUS consists of one RISC processor and 512 reconfigurable PEs (Processing Elements). The PEs have coarse-grained reconfigurable data-paths and hierarchical 2-D reconfigurable interconnections, which can be dynamically configured into several independent groups to perform parallel processing for different computing-intensive tasks. Furthermore, fast configuration and data buffering techniques are utilized to promote the overall throughput of the whole system [7]. It's an assumption target of the simulator.

The REMUS processor consists of an ARM11, two PEAs, an Entropy Decoder (EnD), and some assistant modules, such as an interrupt controller, a DMA (Direct Memory Access) controller and an AXI2AXI bridge illustrated in Fig. 1. An ARM11 is a typical embedded RISC used for application control and reconfigurable schedule, with two Tightly Coupled Memories to accelerate the specific loaded codes. An EnD is a configurable stream decoder, which enables high performance on entropy decoding such as CAVLD (Context-Adaptive Variable-Length Decoding) and CABAD (Context-Adaptive Arithmetic Binary Decoding). A PEA is a powerful dynamic reconfigurable system consisting of 256 processing elements. Algorithms can be

Fig. 2 The PEA architecture.

mapped into a PEA at the same time and run independently to achieve high performance just like ASICs. The details of the PEAs will be discussed next.

2.1 Processing Element Arrays

Each group of PEAs provides efficient datapaths and I/Os with 256 reconfigurable cells and routes that can be refunctionalized and restructured through the context interface. The schematic of the PEA is illustrated in Fig. 2. Here, $PE8 \times 8$ is the basic unit, defined as the minimal function block, which has three parts: DBI, FCI, and $PE8 \times 8$. The DBI is a flexible data exchange unit with asymmetric FIFOs. It can prepare data from DDR2 memories, internal memories, and intermediate results. The FCI takes charge of controlling flow. It is fixed by the ARM or the DMAs before calculations, and it is altered during the PEs' processing. Context information stored in it supports different kinds of applications, which could be updated from time to time; this is the key to REMUS' fast updating. $PE8 \times 8$ is a coarse-grained reconfigurable computing array of 64 16-bit cells, and routes consist of a reconfiguration network. Every unit has DBI, FCI, and $PE8 \times 8$.

Each PE adopts the common ALU architecture [8] with arithmetic operations (addition, subtraction, multiplication etc.) and logical operations (bitwise logic operations, bit-shifting operations) supported, as illustrated in Table 1. Different kinds of heterogeneous cells should be adopted so as to satisfy the requirements of various applications. Cell array also consists of TRAs (Temp Register Arrays), the grass-green square array in the middle of $PE8 \times 8$ illustrated in Fig. 2. They can transfer data, make up pipeline, and be assembled into array extension interface. The granularity and routing style are flexible.

Table 1 Functions of PEA.

$A+B$	$A \wedge B$	$A \geq B$	$A+(B \gg C)$	$(A-B) \gg C$	$A \times B_L$
$A-B$	$A \sim B$	$A \leq B$	$A+(B \ll C)$	$(A-B) \ll C$	$A \times B_H$
A	$A > B$	$ A-B $	$A-(B \ll C)$	$A-(B \gg C)$	--
$\sim A$	$A < B$	$C ? A : B$	$(A \gg C) - B$	$(A \ll C) - B$	--
$A \& B$	$A == B$	$A \ll B$	$(A+B) \gg C$	$\text{Clip}(A, -B, B)$	--
$A B$	$A != B$	$A \gg B$	$(A+B) \ll C$	$\text{Clip}(A, 0, B)$	--

Fig. 3 The configuration architecture.

2.2 Configuration Architecture

Context, which defines the functions of total PEAs, is divided into four parts: load FIFO configuration, store FIFO configuration, PEA configuration, and customization. Load FIFO configuration and store FIFO configuration decide the data fetching styles of DBIs (discussed in the previous section). PEA configuration defines the functions of processing cells and the connections of data-paths. The IO addresses and fetching styles of the DBI could be modified through customization. This last part makes it possible to change configurations of the PEA quickly, which could be done by ARM outside or by the PEA itself.

Contexts are first moved into context memory with a mark number for each. ARM then arranges the mark numbers into the FIFO to describe the control flow of the PEA. After that, the PEA is turned on automatically; the loading, calculating, and storing processes are executed in parallel, as shown in Fig. 3. PEA has a particular path for inserting context mark numbers into the FIFO, which builds up a flexible context switching mechanism and makes complex algorithms' mapping much more convenient.

Throughput computing is brought in here. Configurations and calculations are all IO-dependent. Throughput computing requires the loading/storing process to be busy at

Fig. 4 The operation queues of PEA.

all times and the IOs to be full, as illustrated in Fig. 4. In a full-throughput situation, statistics of the efficiency of calculations on the PEA help us tailor the size of PEA units, such as $PE8 \times 8$, which has suitable processing power to fill the calculation phase's gaps in the H.264 decoding application. Reducing areas do not affect the performance of REMUS in this way. So the reconfigurable elements' size can be optimized for applications.

3. SimREMUS Architecture

3.1 Cycle-Accurate Mechanism

SimREMUS supports embedded software and signal level observation at the same time. So the cycle-accurate figure is necessary.

Cycle accurate simulation is based on a two phases iteration — update and communicate [9]. Every module is function modeling, and connected by transaction level interface. Each period, modules get their inputs first in the communicate phase and then do the calculation at the update step, which is illustrated in Fig. 5. During communicate steps, the registers of REMUS get their new values from the front transmission logics; while during update steps, the new values of the registers are used to calculate new results on the back transmission datapaths. Communication steps are triggered by the posedge of the clock, and update steps follow the corresponding communication steps. All the valuable signals such as interfaces of PEA and configuration registers are described like this, so they could be observed on a cycle-accurate level.

Simulation of inner part is based on behavior modeling in SystemC. Modules and interfaces are all classes, which is easily inherited and polymorphed. It's shown as Fig. 6. Inter-connection classes describe buses, wires and specific signals, module classes are system level abstracted, fast constructed, and fast simulated. Debug and profile functions are also packaged into separate classes, transferring information between log files or interactive interfaces.

3.2 PEA Simulation Model

PEA modeling adopts Object-Oriented concept, and packages sub-modules into classes, which is shown as Fig. 7. The small squares are processing elements; the big one is

Fig. 5 The simulation architecture.

Fig. 6 Simulator modularization.

Fig.7 PEA simulation model.

a reconfigurable network defining routes. The grey columns are configurable memories [10].

The granularities, functions of PE are determined by a common definition file. Using Typedef to change IO types makes granularities easily to be tested. PE functions are written in case statements. PE transmits and receives data through the reconfigurable network which is an all-connecting style. All these reconfiguration figures are abstracted for design iteration.

Using the cycle-accurate mechanism as mentioned in 3.1, transactions at the interface can be observed, but there is not enough. So an inner events recording is designed. Function choices, connects and step control information are totally written in log files.

Fig. 8 System integration of SimREMUS.

Fig. 9 Example of slave (left) and master (right) port implementation.

Fig. 10 Simulation waveforms.

3.3 REMUS System Level Integration

SoCdesigner is adopted here to integrate the whole system, since it is providing ARM cycle-accurate simulation models. DMA, Interrupt controller, profiling module, kinds of memories, PEAs and ARM core are integrated through buses. There are two kinds of buses, one is AMBA2.0, and the other is a specific local fast bus. Master ports are simple, which drive sizes, burst type and address, etc. Slave ports is a class packaging two main functions, read() and write() (Fig. 9). Profiling on the bus provides waveforms (Fig. 10) [9].

3.4 System Profiler

SimREMUS implements system profiling in two kinds of ways, software profiler and hardware profiler.

Fig. 11 (a) Information data stream structure, (b) Context data stream structure.

• Software Profiler

Software profiler is used in applications, for ARM to record customized information such as current cycles, tracing variables, debug signals, etc. it's implement in a simulation module on the platform, called profiler module in Fig. 8. Information types are customized in a kind of sub-script, which be loaded into profiling module at the beginning. Profiling module is allocated in certain addresses, to which software writes records.

• Hardware Profiler

Hardware profiler is implemented in every module, which is shown in Fig. 6. Each module has a Profiling Interface (PI). PI collects hardware information during the simulation phase, as illustrated in Fig. 11. Different kinds of data type are supported and organized in streams or channels. Streams are divided into information data stream and context data stream. Former one records headers and latter one collects results.

3.5 Power Analysis

Power is a critical concern in many applications, which should be considered in this simulator. The system power consumption is divided into four parts, memory accessing power, PEA computing power, ARM core power and the others' part.

Memories cost most power. Analysis on separate operations comes out first and then profile operation counts [11]. Define the voltage, frequency, and derive the power consumption at last. PEA power consists of register switching, data-path consuming and temp/context memory accessing [12]. The power estimation is based on operation

Fig. 12 Example of hardware profiler.

Fig. 13 Power analysis of memories.

statistics, such as the number of calculation operations, register toggles and memory accesses etc. Since the energy consumption of each device can be located by foundry documents (using TSMC 65 nm CLN65G+ HVT Process here), which can be easily converted to the estimated value on specific applied voltage and frequency according to the generic dynamic power consumption equation. The profiled power estimation has more relative meanings rather than actual meanings, which can be used to compare the quality of two architectures from power perspective. Figure 12 and Fig. 13 illustrate results of power analysis.

3.6 Fast Simulation Technology

Except function implementation, speed may be the most important issue of a simulator. SimREMUS gains a very fast simulation nearly 300 kcycle per second, through several approaches.

• SystemC Data Type Replacing

SystemC data type contains complex methods and definitions, which costs extra code size and slows executing efficiency. So redundancy variables should be simplified as follows.

```
sc_uint<32>      ->    unsigned int
reg.rang(17, 16) ->    reg>>16&0x3
```

This approach earns 2–3 times speed-up.

Fig. 14 Conditional calculation.

• Conditional Calculation

The simulation of the module under test could be skipped when it is predicted that the state transitions of the input signals and the FSM (Final State Machine) will definitely not occur as shown in Fig. 14. This depends on how to detect the variations of the critical input signals (e.g. ENABLE, CS signals etc.) and state registers reliably and efficiently. In the proposed approach, the values of the critical input signals and internal state registers of each component in previous simulation stage are preserved so as to serve as comparison references in current stage. For example, when running simulation of the “PEA” module, if it is predicted that the status of the input signals and the state registers of the “Array” component will not change, the simulation engine will directly advance to next component, leaving the “Array” component un-simulated. Relying on the reduction of the meaningless calculation, the performance of the simulator will be substantially improved.

This approach earns about 20x simulation performance boost according to the measurement.

The key note here is to reduce executing codes as soon as possible. Hardware actually calculates every moment, but the simulator doesn’t. Although cycle-accurate issue needs signal level information to make system complicated, condition driven modules could also be described for fast simulations.

4. REMUS Compiler

Compiler is the most important supporting tool to facilitate the use of reconfigurable computing architecture. A template-based compiler framework for reconfigurable computing architecture is presented as Fig. 15 [6]. The compiler’s input is the application source code of native high-level programming language, e.g. C language. The compiler synthesizes the executable supporting run-time reconfiguration for PEA. The reconfiguration points and contexts are embedded in the executables such that the PEA is reconfigured along with the executables are running. Instead of the operator-based synthesis in previous reconfigurable computing compilers, the template-based synthesis algorithms are used to improve the execution performance and reduce the configuration context size. At compile-time, the compiler automatically analyzes the source code and extracts several operation-templates which occur most frequently. By using these templates in configuration context synthesis, the number of intermediate registers between operations can be

Fig. 15 (a) The block diagram of compiler, (b) The front-end processing flow.

Fig. 16 (a) Template examples, (b) An overview of the basic template matching algorithm.

reduced drastically which results in small configuration context size and short reconfiguration time.

Back-end compiler contains template extraction, Template Matching and Context Generation and Patching. The architecture of PE is feasibly used for combining some PEs into a computing entity (called template) for reducing intermediate registers between operations. The use of templates can significantly improve the computation performance of PE array. Furthermore, since the templates are the frequently used sub-graphs in the application, the configuration context of one template can be reused multiple times. Then the time used for transferring context to context buffer and the reconfiguration time are dramatically decreased. Template matching is to select templates from pre-defined and extracted template set to cover the DFG (Data Flow Graph) nodes and data edges. Template examples and an overview of the basic template matching algorithm are illustrated in Fig. 16.

The algorithm starts by calling the function *EvaluateCoverage* to compute the coverage of each template in template set (predefined and extracted) and sorting the template based on the coverage value by using the

function *SortTemplate*. Based on the sorting results, to select the template for matching function *select_template* is performed to select the template from the template set and compute the coverage of the input DFG graph. Finally, the function *exit_conditions* is called to halt the selecting procedure. The function returns “TRUE” (exit the loop) if the DFG graph is sufficiently matched (covered) or there is no suitable template to select.

According to matching results, the matched templates will be assigned to map and schedule on reconfigurable hardware to achieve high performance and generate the optimal configuration context.

To verify and evaluate the proposed compiler framework, the compiler is implemented with the hardware architecture of REMUS. SUIF2 (Stanford University Intermediate Format compiler) and MachSUIF tools are leveraged for some standard optimizations. And some SUIF2 passes are implemented for front-end processing. The back-end processing is described in C. This compiler synthesizes real executable of H.264 decoder for REMUS and evaluate the performance improvement. DCT and Motion Estimation (ME) as test cases are given to evaluate the performance of our compiler. The comparison of equivalent execution time cycles on different platforms are shown in Fig. 17. The

execution of binary code compiled by the proposed compiler is much faster than PipeRench, Morphosys, and TI DSP (TMS320DM642) [10]. Since REMUS, PipeRench and Morphosys have similar PE array structure, the performance gain is mainly from the compiler optimization, especially from template-based synthesis. The template-based synthesis can fully utilize the resource of PE array and reduce the number of intermediate registers operation, which speeds up the execution speed. Execution time of compiler-generated executable is compared with the manually-generated executable, as shown in Fig. 17. It can be seen that the performance of compiler-generated executable is approaching manually-generated executable (labeled “REMUS with full-custom”). Considering the extreme complexity of manual generation, the proposed compiler is good enough for generating executable for the proposed reconfigurable processor.

5. System Debugger

SimREMUS also provides a strong debugger. Every module contains a Debug Interface (DI), which is shown as Fig. 6. The system debugger accesses every module through DI. During the simulation contexts of memories and registers can be displayed dynamically, and overwritten, too. Breakpoints to any signals/memories and step-simulation are supported.

The system debugger is divided into five parts, simulation host, debugger, DI dispatcher, DI target and SI model. The executing flow is illustrated in Fig. 19. Simulation host takes charge of communicate/update iteration to emulate cycles’ action. Debugger traces signals, triggers break processing. Dispatcher pipelines commands between the host and debugger, DI target.

Fig. 17 (a) 100 times of 4 × 4 DCT, and (b) 100 times of 16 × 16 ME.

Fig. 18 The mapping sketch of HD 1080p decoding application.

Fig. 19 Debugger mechanism.

6. Case Study

REMUS processor is designed by the proposed SimREMUS. The REMUS processor targets at a variety of media processing applications, such as H.264 [13], AVS [14], MPEG-4, etc. So far, because of the very high computing workload, 1080p@30 fps HiP (High Profile) decoding is still not reachable in the reconfigurable processor. But through current section this computing-intensive algorithm is mapped and efficiently executed in REMUS.

In order to achieve H.264 HiP 1080p@30 fps decoding, 8100 MBs (1920*1088/256) are needed, executed in 6.67 million clock-cycles (200/30) when exploiting a 200 MHz system clock. Only 850 clock-cycles (6.67 M/8100) can be allocated to the processing of each MB. Therefore, the parallel processing among ARM11, PEAs, and the EnD module is very pivotal. From a system level perspective, there are four pipelined stages (shown in Fig. 18.): ① neighbor computing on ARM11; ② entropy decoding on EnD; ③ data/context preparing on ARM11; and ④ MB decoding on the PEAs. Stages 1, 2, and 3 are executed in 850 clock-cycles, but stage 4, which consists of all the computing-intensive algorithms, such as intra/inter predictions, IQT (Inverse Quantization and Transform), and deblocking, requires a little more. Therefore, two PEAs have to be utilized to run in a ping-pong mode. In this way, one MB can be decoded in four units (850-cycle is set to be a time unit.). When the pipeline stages are all full, each MB shares one unit on average.

The frequently used characteristics of REMUS are illustrated in Table 2, which is decided during the simulation. The set of criteria is defined in [15]. H.264 decoding speed on SimREMUS is shown in Table 3.

SimREMUS showed that H.264 HiP decoding @Level4 could be achieved when exploiting a 200 MHz working frequency on REMUS; the performance was 92.5% faster than that of XPP as shown in Table 4. It implemented into $3.4 \times 3.4 \text{ mm}^2$ silicon by SMIC's 65 nm logic process with a 400 Mhz maximum working frequency. The silicon area of REMUS is 2.2 times smaller than XPP when normal-

Table 2 Characteristics of REMUS determined on SimREMUS.

Granularity	Programmability	Reconfiguration	Sub-Array size
16bits	Multiple	dynamic	8x8
PE number	Route	heterogeneous	Application Domain
512	2-D local-connect	Yes, multiply, ram, alu	Media/baseband app.

Table 3 H.264 decoding simulation performance on SimREMUS.

Stream	foreman	mobile	Van.Helsing
size	qcif	cif	1080p
Profile	Main	main	high
frames	300	30	10
speed	271kcycle/s	259kcycle/s	270kcycles/s
time	11.76s/f	49.3s/f	983.3s/f

Table 4 Comparison among REMUS and the other H.264 decoders.

	Intel P4[16]	ASIC[17]	XPP[4]	ADRES[18]	REMUS
Frequency	2.4Ghz	120Mhz	450Mhz	300Mhz	200Mhz
Profile	BP	BP	BP	BP	HiP
Max Spec.	1280*720 @12fps	1280*720 @30fps	1920*1080 @24fps	1280*720 @30fps	1920*1080 @30fps

Table 5 Silicon implementation comparison.

	XPP[4]	ADRES[18]	REMUS
Process	90nm	90nm	65nm
Area	75mm ²	64mm ²	11.6mm ²
Max Freq.	450Mhz	300Mhz	400Mhz

ized to the same manufacturing process which is illustrated in Table 5.

7. Conclusion

This paper derives a cycle-accurate simulator for a reconfigurable multi-media system, called SimREMUS. SimREMUS supports high-level representations to model and cycle-accurate simulation the reconfiguration. Many important issues of reconfigurable system, such like granularity, heterogeneous processing elements, different routing can be easily evaluated. Compiler and debugger are also developed, making up of a complete tool chain. SimREMUS simulated 270 kcycles per second for this millions gates system and produced one 1080p frame only in 15 minutes. Using SimREMUS a reconfigurable architecture is implemented and result showed that 1080p@30 fps of H.264 HiP@ Level 4 can be achieved on final architecture. More trade-off options should be evaluated in this simulator as a further work.

Acknowledgements

This work was supported by NNSF of China grant 60803018 and the National High-Technology Research and Development Program ("863" Program) of China grant

2009AA011702.

References

- [1] A.V. Brito, M. Kuehnle, E.U.K. Melcher, and J. Becker, "A general purpose partially reconfigurable processor simulator (PReProS)," IEEE International, Parallel and Distributed Processing Symposium, IPDPS 2007, pp.1–7, 2007.
- [2] T.J. Todman, GA. Constantinides, S.J.E. Wilton, O. Mencer, W. Luk, and P.Y.K. Cheung, "Reconfigurable computing: Architectures and design methods," IEE Proc. Computers and Digital Techniques, vol.152, pp.193–207, 2005.
- [3] S. Fytraki and D. Pnevmatikatos, "ReSim, a trace-driven, reconfigurable ILP processor simulator," Design, Automation & Test in Europe Conference & Exhibition, pp.536–541, 2009.
- [4] M.K.A. Ganesan, S. Singh, F. May, and J. Becker, "H. 264 decoder at HD resolution on a coarse grain dynamically reconfigurable architecture," Field Programmable Logic and Application, FPL2007, pp.467–471, 2007.
- [5] <http://www.arm.com/products/DevTools/RealViewSystemDevelopment.html>
- [6] C. Yin, S. Yin, L. Liu, and S. Wei, "Compiler framework for reconfigurable computing architecture," IEICE Trans. Electron., vol.E92-C, no.10, pp.1284–1290, Oct. 2009.
- [7] M. Zhu, L.B. Liu, S.Y. Yin, Y.S. Wang, W.J. Wang, and S.J. Wei, "A reconfigurable multi-processor SoC for media applications," Proc. IEEE International Symposium on Circuits and Systems, ISCAS 2010, pp.2011–2014, 2010.
- [8] S.D. Yu, L.B. Liu, and S.J. Wei, "Design of reconfigurable processor based on the loop mapping," Journal of Beijing University of Posts and Telecommunications, vol.32, pp.9–14, Aug. 2009.
- [9] http://infocenter.arm.com/help/topic/com.arm.doc.dui0052c/ADP_ARM_DUI0052C.pdf
- [10] P. Baglietto, M. Maresca, and M. Migliardi, "A simulator for reconfigurable massively parallel architectures," Proc. Second Euromicro Workshop on Parallel and Distributed Processing, pp.185–189, 1994.
- [11] L. Ming, Y. Jun, and Z. Yongxing, "SDRAM power model and instruction FIFO optimization," Electronic Components and Materials, vol.28, pp.834–838, 2005.
- [12] X. Wei, Z. Binyu, and Z. Chuangqi, "Architecture-level power modeling and analyzing for reconfigurable processor," Computer Engineering and Applications, vol.43, pp.34–37, 2007.
- [13] T. Wiegand, et al., "Overview of the H.264/AVC video coding standard," IEEE Trans. Circuits Syst. Video Technol., vol.13, no.7, pp.560–576, 2003.
- [14] "Information technology — advanced coding of audio and video — Part 2: Video," Chinese Standard Press, GB/T 20090.2-2006, 2006.
- [15] H. Singh, M.H. Lee, GM. 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.
- [16] X. Zhou, E.Q. Li, and Y.K. Chen, "Implementation of H. 264 decoder on general-purpose processors with media instructions," SPIE and IS&T 2003, pp.224–235, 2003.
- [17] T. Chen, S. 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, 2006.
- [18] B. Mei, F.J. Veredas, and B. Masschelein, "Mapping an H. 264/AVC decoder onto the ADRES reconfigurable architecture," FPL2005, pp.622–625, 2005.

Min Zhu was born in 1984. He 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.

Shouyi Yin received the B.S., M.S. and Ph.D. degrees in Electronic Engineering from Tsinghua University, China, in 2000, 2002 and 2005 respectively. He has worked in Imperial College London as a research associate. Currently, he is with Institute of Microelectronics at Tsinghua University as an assistant professor. His research interests include mobile computing, wireless communications and SoC design. Dr. Yin has published more than 20 refereed papers, and served as TPC member or reviewers

for the international key conferences and leading journals.

Chongyong Yin received the B.S. degree from the Department of Microelectronic, Xi'dian University, Shanxi, China, in 2006. Currently, he is working toward the Ph.D. degree in the Institute of Microelectronics, Tsinghua University, Beijing, China. His research interests include reconfigurable computing and its compiler design.

Shaojun Wei was born in Beijing, China in 1958. He received Ph.D. degree from Faculte Polytechnique de Mons, Belgium, 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.