IEIE Transactions on Smart Processing and Computing, vol. 4, no. 2, April 2015
http://dx.doi.org/10.5573/IEIESPC.2015.4.2.083

Core-A: A 32-bit Synthesizable Processor Core

Ji-Hoon Kim[1,*], Jong-Yeol Lee[2], and Ando Ki[3]

[1] Department of Electronics Engineering, Chungnam National University / Daejeon, South Korea jihoonkim@cnu.ac.kr
[2] Division of Electronic Engineering, Chonbuk National University / Jeonju, South Korea jong@jbnu.ac.kr
[3] R&D Center, Dynalith Systems / Daejeon, South Korea adki@dynalith.com

***** Corresponding Author: Ji-Hoon Kim

Received November 20, 2014; Revised December 27, 2014; Accepted February 12, 2015; Published April 30, 2015

** Short Paper*

Review Paper: This paper reviews recent progress, possibly including previous works in a particular research topic, and has been accepted by the editorial board through the regular review process.

***Abstract*:** Core-A is 32-bit synthesizable processor core with a unique instruction set architecture (ISA). In this paper, the Core-A ISA is introduced with discussion of useful features and the development environment, including the software tool chain and hardware on-chip debugger. Core-A is described using Verilog-HDL and can be customized for a given application and synthesized for an application-specific integrated circuit or field-programmable gate array target. Also, the GNU Compiler Collection has been ported to support Core-A, and various predesigned platforms are well equipped with the established design flow to speed up the hardware/software co-design for a Core-A–based system.

***Keywords*:** Processor core, CPU, Embedded processor, Development environment

1. Introduction

As the market for embedded systems greatly increases, system-on-chip (SoC) plays a key role, and the importance of an embedded processor core also increases. Although instruction set architectures (ISAs) like ARM, IBM, and Intel have enjoyed huge success, the huge cost for licenses rules out academia and others working with a small volume of equipment [1]. There have been several royalty-free processors, but the internal structure is too complicated or mainly targets high-performance applications [2, 3]. Although there have been various evolutions in ISA over the past 20 years, after the debut of the ARM7 central processing unit (CPU), there is still huge demand for a low-complexity synthesizable processor core [4].

In this paper, we present the Core-A processor, which was developed by the Korea Advanced Institute of Science and Technology (KAIST) in 2007. It has a unique instruction set architecture, and the development environment is well established by porting the GNU Compiler Collection (GCC) and by including an on-chip hardware debugger. Its five-stage implementation can run at more than 200MHz and requires just 32K logic gates when fabricated in 130nm and 180nm complementary metal-oxide semiconductor (CMOS) processes [5]. It was designed using Verilog-HDL, and the source code is in the public domain and available without royalty. Also, there is well-established hardware/software (HW/SW) co-design flow developed by Dynalith Systems [6].

2. Core-A Instruction Set Architecture

As illustrated in Fig. 1, Core-A has 16 general purpose registers (GPRs), two processor modes, and seven precise exception types. Among the 16 GPRs, three are allocated for predefined behavior, such as return address, save for, exception, and function call. Also, it has two special purpose registers: PC (Program Counter) and PS (Program Status). All instructions are 32 bits long and have a simple encoding map for lower decoder complexity. Like many Reduced Instruction Set Computer (RISC) designs, it is a “load-store” architecture where the only instructions that access main memory are loads and stores [7]. All arithmetic and logic operations occur between internal registers. Also, it can have up to four coprocessors, such as a floating-point unit, where each coprocessor can have up

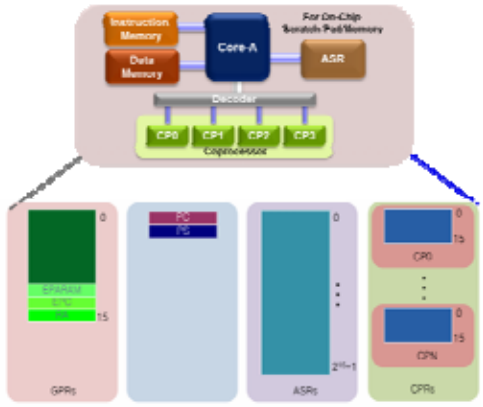


Fig. 1. Core-A Architecture – Programmer's View.

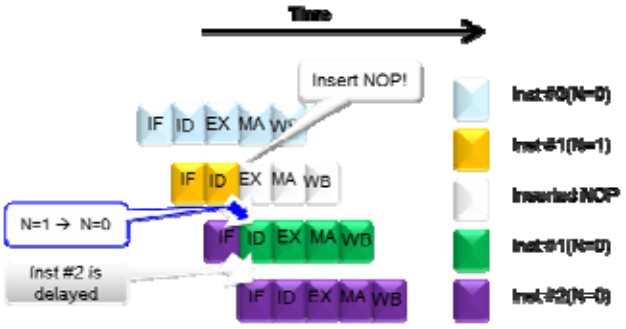


Fig. 2. NOP insertion by '*N*-bit'.

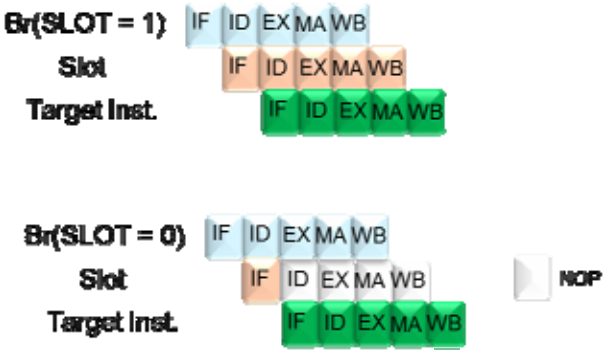


Fig. 3. Programmable delay slot.

to 16 registers, and additional memory space (up to 64KB), so-called ASR (Application Specific Register), can be included where frequently used data can be stored for efficient processing. Two special instructions—MTA (move to ASR) and MFA (Move from ASR)—are used for data transfer between ASR and general purpose registers. Since the address space for ASR is not included in main memory, ASR can be used as on-chip scratch pad memory, which is usually well exploited in DSP (Digital Signal Processing) applications. Also, it can be used as the communication channel between Core-A and an application-specific hardware accelerator in which internal registers can be efficiently accessed.

For code density and DSP applications, Core-A has several unique features. First of all, a pre-shifted operand is supported in arithmetic/logic unit (ALU) instructions, which can reduce the code size efficiently. Also, multiple load/store instructions, which can move up to 16 data, are supported. Additionally, powerful and various addressing modes in load/store instructions can reduce the number of instructions required in memory address calculation.

Due to the data dependency that frequently occurs in pipelined implementation, recent embedded processors insert the explicit No-Operation (NOP) instruction to resolve the dependency, which leads to code size increase [7]. For a MIPS processor, the average amount of NOP instructions is about 10% [5]. Also, many RISC designs include a branch delay slot, a position after a branch instruction that can be filled with an instruction that is executed regardless of whether the branch is taken or not. Although this feature can improve performance of pipelined processors, it can complicate both multi-cycle CPUs and superscalar CPUs [3]. Accordingly, in the Core-A ISA, the number of delay slots is defined as programmable. In Core-A, the large amount of explicit NOP instructions can be avoided via two special features: '*N*-bit' in all instructions, and programmable delay slot in branch instructions.

- '*N*-bit' in all instructions means insert NOP automatically before execution of this instruction, as shown in Fig. 2.
- Branch SLOT field indicates the number of instructions to be executed after the Jump/Branch instruction, which is taken as shown in Fig. 3.

In pipelined implementation of the processor, these two features can be exploited effectively to reduce the number of explicit NOP instructions, which are mainly used for resolving dependency problems.

In addition, as the importance of an efficient signal processing capability increases in the embedded processor, the Core-A ISA supports the following instructions.

- Signed/Unsigned Long Multiply
- Signed/Unsigned MAC (Multiply-Accumulation)
- Various bit-wise manipulation instructions
- the Move Upper Immediate (MUI) instruction for 32-bit immediate loading

For performance evaluation, the CoreMark benchmark was exploited, which is designed specifically to test the functionality of a processor core [8]. With the GCC-based Core-A compiler, about 37% of the instructions exploit '*N*-bit,' and CPI (Cycles Per Instruction) is measured as 1.62 with the –O2 option in its five-stage implementation., which is comparable to ARM9E-S and ARM7TDMI [9].

3. Core-A Hardware Implementation

Core-A was implemented with five-stage pipelines – IF (Instruction Fetch), ID (Instruction Decode), EX (Execution), MEM (Memory Access), and WB (Write Back). Due to the orthogonality between the datapath and controller in the implementation, the gate count and the performance are well balanced. The Core-A processor was described in Verilog-HDL and is fully synthesizable. It has

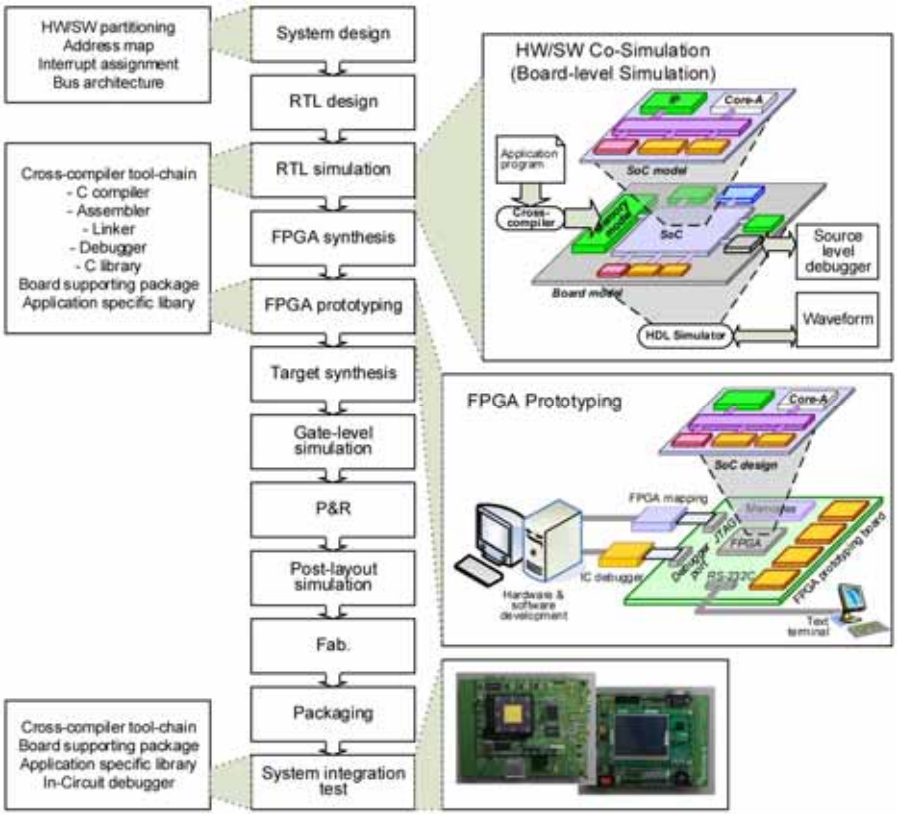


Fig. 4. Core-A system design and verification flow.

two memory interfaces: one for instruction fetch and another for data memory access. Since they are designed to support synchronous SRAM timing, an additional wrapper can be attached to support various on-chip interconnection protocols, such as AMBA AHB/AXI. Also, according to the specification, the memory management unit (MMU), memory protection unit (MPU) and cache can easily be attached without changes in Core-A. Additionally, Core-A is designed to easily support IP-level clock gating for low-power consumption. Its five-stage implementation can run at 260MHz and requires just 32K logic gates when fabricated in a 180nm CMOS process [5]. It occupies only 0.47mm^2 and its power efficiency is about 0.35mW/MHz. Core-A can run faster and requires less power consumption if fabricated in the advanced CMOS process. Several implementations based on the Core-A processor have been introduced [10, 11].

4. Core-A Development Environment

The system embedding Core-A processor is recommended to follow the HW/SW co-design flow illustrated in Fig. 4, which includes system design, building hardware, preparing software, HW/SW co-simulation, FPGA-based prototyping, silicon fabrication, system integration, system testing, and system maintenance. In this section, hardware development, software development, and software debugging features are summarized.

4.1 Hardware Environment

In the system, the Core-A processor controls all hardware components, which should be tested by the software. Accordingly, HW/SW co-simulation, which is also known as board-level simulation, becomes the key part of hardware design, as shown in Fig. 5. In HW/SW co-simulation, several important issues, such as system connection, address partitioning, interrupt assignment, multi-byte data ordering, and frequency-dependent parameters, should be checked and verified in terms of both hardware and software.

4.2 Software Environment

The software tool chain for Core-A, illustrated in Fig. 6, was developed by porting GCC, which includes a compiler that translates high-level C language into assembly code; assembler, which translates assembly code into binary machine language; and various utilities, which can manipulate binary files, the so-called binutils.

The development of a compiler for the new processor is well known as compiler porting for the new target machine, which requires the modification or description of the GCC backend. For successful compiler porting, the following issues were carefully considered.

- Accurate and adequate information for the target machine is necessary because code generation highly depends on the ISA of the target machine.
- All information for code generation should be well described in two files: *target description macro*,

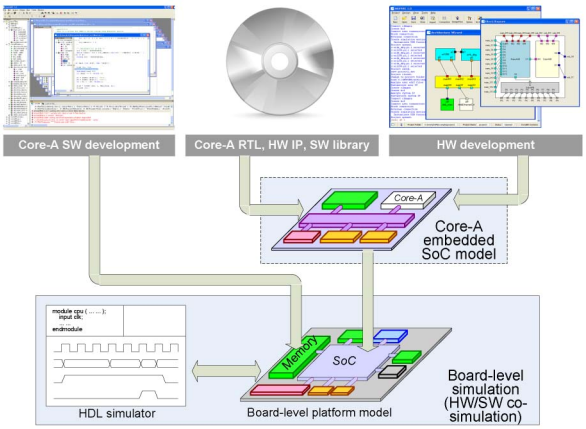


Fig. 5. Core-A system development environment.

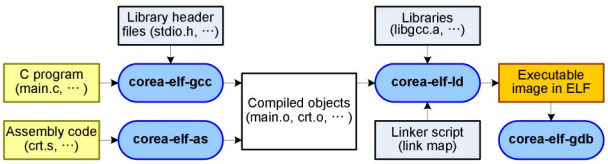


Fig. 6. Core-A software tool chain.

which denotes the hardware aspects of the target machine, and *machine description*, which provides information regarding the types of instructions and their behavior.

Also, a cycle-accurate instruction set simulator with various debugging features, such as setting break-points and monitoring, was created for SW development.

4.3 Debugging Environment

In addition to the cycle-accurate instruction set simulator, for more powerful debugging features, GDB (The GNU Project Debugger) was also ported to the Core-A processor as a source-level debugger. GDB consists of three major parts: a user interface, symbol handling, and target system handling. To port GDB to a new processor, target system handling, which consists of execution control, stack frame analysis, and physical target manipulation, should be newly described. It also defines the kinds of machine-language programs GDB can work with, and how it works with them, which is described in a number of C macros. Usually, debugging is performed based on communications between a host and a target machine, as illustrated in Fig. 7. GDB supports a serial protocol for code that runs on the target machine and provides several sample stubs that can be integrated into target programs or operating systems for this purpose.

There are also two versions of a hardware on-chip debugger (OCD) for the Core-A processor. A JTAG-based debugger exploits boundary scan chains to control the Core-A processor [10]. Another uses a dedicated control path of the Core-A processor and a serial peripheral interface port for an external interface [12]. The Core-A OCD provides following features.

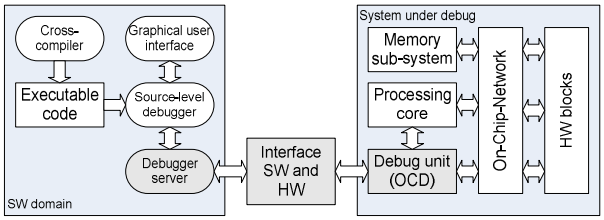


Fig. 7. Core-A Debugging Environment.

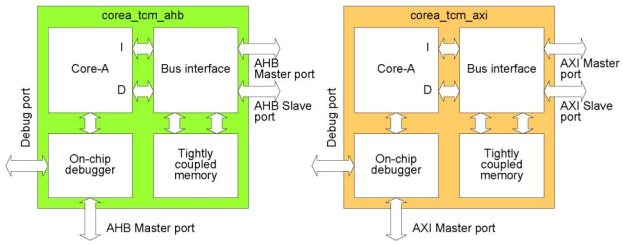


Fig. 8. Core-A processor with AMBA Interface.

- BREAK case I: Stop the Core-A processor just before the instruction at the specific memory address is executed.
- BREAK case II: Stop the Core-A processor just before the specific instruction is executed
- WATCH case: Stop the Core-A processor just after writing data to the specific memory address.

Core-A debuggers also support the GDB Remote Serial Protocol, which is an ASCII message-based protocol to read and write registers and memory over serial communications lines.

5. Core-A Example Systems

For easy integration in SoC, two versions of Core-A that support AMBA AHB and AXI on-chip interconnect are usually adopted, as shown in Fig. 8. For low hardware complexity, those two versions also embed tightly coupled memory (TCM) rather than on-chip cache.

Although the processor is a block system from the hardware point of view, it plays a major role while the system is working since it initializes the system, controls all components, interacts with external activities, and so on. The processor requires memory to store and run programs, the system bus to communicate with other hardware components, and legacy components to support operating systems of a real-time kernel. Those components are the base of the system and its structure varies with the applications. Various platforms based on Core-A were prepared to speed up system design.

- RT-Kernel supporting platform: minimum structure to port real-time kernels consisting of serial communications, interrupt controllers, tick-timers, and memory.
- Audio platform: audio controller added.
- Video platform: LCD controller added.

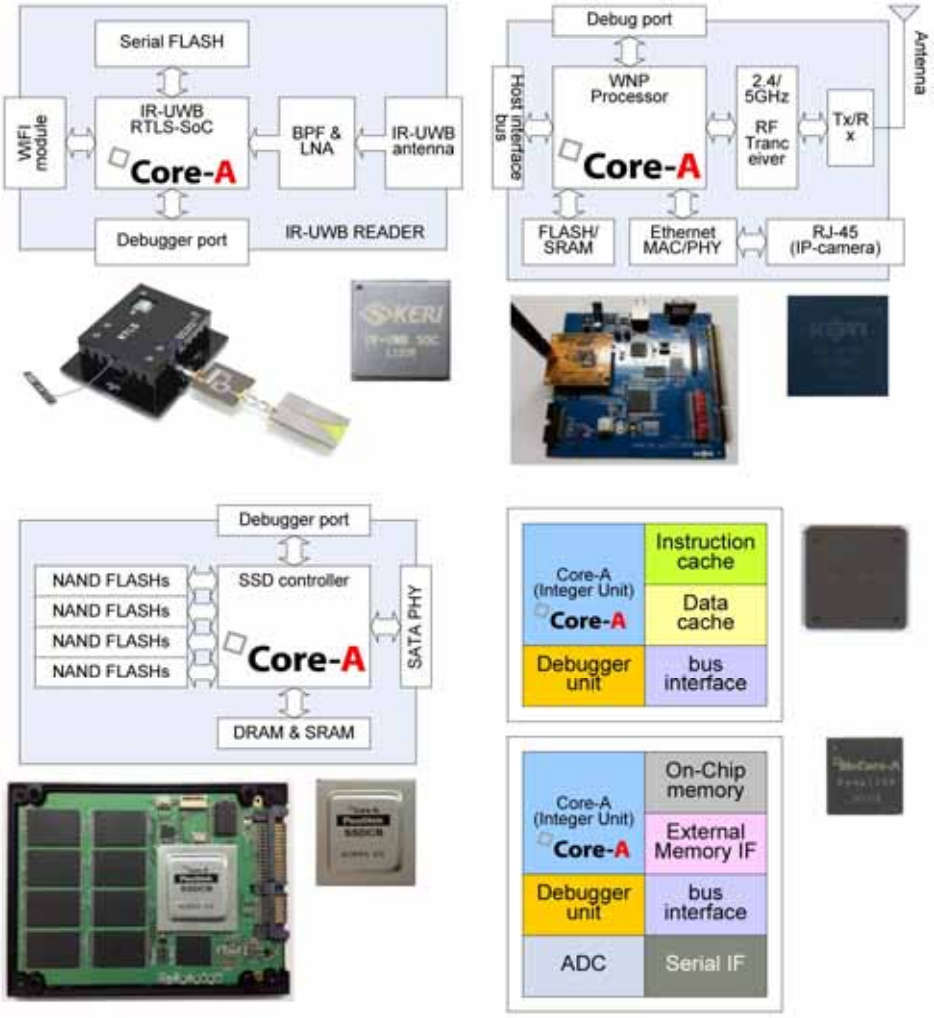


Fig. 9. Various application System-on-Chips based on Core-A.

- Camera platform: image sensor controller added.
- Network platform: Ethernet MAC added.

There are several implementations, as illustrated in Fig. 9, which include communications, storage controllers, micro-controllers, and so on.

6. Future Roadmap

As the importance of the Internet of Things (IoT) and wearable computing increases, small and ultra-low-power 32-bit processors become necessary. To improve the processing efficiency of Core-A, a micro-architecture such as the cycle-behavior of instructions and the number of pipeline stages and branch shadows, can be revised, as were introduced elsewhere [7, 13]. Also, for advanced power management, several instructions can be added and a latch-based pipeline, which is more robust to aggressive voltage scaling, can be exploited.

In addition, the performance of the Core-A compiler can be increased by changing the code optimization routines. Accordingly, the latest version of GCC or the LLVM Compiler infrastructure, which is a collection of modular and reusable compiler and toolchain technologies, can be exploited [14].

7. Conclusion

In this paper, we have presented a 32-bit processor core called Core-A. In addition to the instruction set architecture and its implementation of the Core-A processor, a development environment (including a compiler tool chain, hardware/software debugger, and several example systems based on the Core-A processor) are introduced. Also, the future roadmap of the Core-A processor is briefly described. Since the Core-A processor is royalty-free, it can be affordable for more devices targeting the Internet of Things, where low cost is necessary.

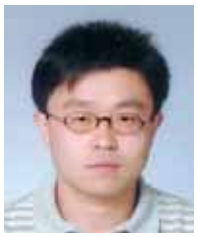
Acknowledgement

Core-A was developed by a group of academic and industry organizations under a project supported by The Korean Intellectual Property Office (KIPO). The project was led by Prof. In-Cheol Park from KAIST.

References

[1] C. Demerjian, "A Long Look at How ARM Licenses Chips," Article (CrossRef Link)
[2] OpenCores, Article (CrossRef Link)
[3] RISC-V ISA, Article (CrossRef Link)
[4] Microprocessor Report, "The Case for Licensed Instruction Sets," The Linley Group, Aug. 18, 2014. Article (CrossRef Link)
[5] Ji-Hoon Kim, et al., "Design of High-Performance 32-bit Embedded Processor," pp. III-54 – III-55, ISOCC 2008. doi: 10.1109/SOCDC.2008.481574, Article (CrossRef Link)
[6] Dynalith Systems Co., Article (CrossRef Link)
[7] Hennessy and Patterson, "Computer Architecture: a quantitative approach," Morgan Kaufmann Publishers.
[8] CoreMark – An EEMBC Benchmark, http://www.eembc.org/coremark
[9] ARM Holdings, Article (CrossRef Link)
[10] Jingzhe Xu, et al., "Design and Verification of Efficient On-Chip Debugger for Core-A," Journal of IEEK – SD, vol. 47, no. 4, pp. 322-333, Apr. 2010. Article (CrossRef Link)
[11] Xuelong Xu, et al., "The Design of Multi-media SoC Platform Based on Core-A Processor," Journal of IEEK, vol. 50, no. 6, pp. 1415-1420, June 2013. doi: 10.5573/ieek.2013.50.6.099, Article (CrossRef Link)
[12] Ando Ki, "Application design using Core-A processor," Hongrung Publishing Co., 2011.
[13] Microprocessor Report, "Casting Off the Pipeline," The Linley Group, Nov. 10, 2014. Article (CrossRef Link)
[14] The LLVM Compiler Infrastructure, Article (CrossRef Link)

Ji-Hoon Kim is an Associate Professor of Electronics Engineering at Chungnam National University, Daejeon, Korea. He received his BS (summa cum laude) and PhD in electrical engineering and computer science from KAIST in 2004 and 2009, respectively. His PhD work focused on design of high-performance CPUs and baseband modems for mobile hand-held devices. In 2009, he joined Samsung Electronics, Suwon, Korea, where he worked on the SoC architecture design for next-generation cellular modems. He is a member of the steering committee of the IC Design Education Center (IDEC) at KAIST. His current interests include embedded processors, low-rate wireless personal area network modems, and ultra-low-power SoC designs for Internet of Things (IoT) devices.



Jong-Yeol Lee received a BS, MS, and PhD in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Republic of Korea, in 1993, 1996, and 2002, respectively. He joined Hynix Semiconductor, Inc., in September 2002. In 2004, he joined the faculty of the department of Electronic Engineering at Chonbuk National University, where he is now an Associate Professor. His research interests include code optimization algorithms for various processors and the design of low-power digital circuits.

Ando Ki is Director of R&D at Dynalith Systems. He received a BS in electronic engineering from Hanyang University in 1986 and an MS in electrical engineering from KAIST in 1988. After studying at KAIST, he worked for ETRI, where he joined shared-memory multi-processor computer projects including cache-coherent snoop cache and system buses. He received his PhD in computer science from the University of Manchester in 1997. After staying in the U.K., he returned to ETRI, where he carried out logically-shared physically-distributed multi-processor computer projects, including the CC-NUMA directory cache. After working for ETRI, he joined Dynalith Systems, where he led product development in the field of logic simulation acceleration using FPGA for SoC design/verification. His current research interests include system-level design and verification, processors for embedded systems, FPGA prototyping, and simulation acceleration.