On-demand design service innovations

S. Shimizu
H. Ishikawa
A. Satoh
T. Aihara

Offering design services for manufacturers of embedded devices has become a very important business, one in which the three leading customer requirements are time to market, integration of leading-edge technologies, and cost reduction; in short, ***on-demand design services.*** *In this paper, we discuss on-demand design service innovations of several types. First, we discuss our unique field-programmable gate array (FPGA)-based system emulation tool. Although embedded systems comprise a wide range of technologies and components, some important technologies and components are common to most embedded systems. Security and communications are two of these, and we have developed offerings in these areas as well. For security, we developed scalable intellectual property macros to meet the requirements for many kinds of cryptographic circuits. These macros can satisfy specific requirements—performance, size of the silicon area, and power dissipation—for many kinds of embedded systems. For communications, we developed an autonomic network configuration tool which allows an end user to avoid the potential frustration of setting up a network connection and which also automatically performs network security tasks.*

Introduction

In this era of pervasive computing, embedded systems are becoming more sophisticated, and their functionality is approaching that of computers. For example, in 1979, cellular phones simply provided wireless voice transmission; now they can be used to make movies via built-in charged-coupled-device cameras, and then transmit them via e-mail. Such phones can also run Java** applications. Because such embedded systems have complicated structures, they require holistic system architectures in which functions are properly layered from the hardware to the application software level, similar to the design of a computer. To achieve required system functions and performance, it is necessary to deal with the interaction of many different parts of the system, and optimization at the system level is critical. Experience has shown that it is quite difficult to develop all of the parts of an entire embedded system within a single company. Thus, many system designers find it advantageous to procure technology and design services from outside providers.

Another reason for the emergence of technology and design services is that the information technology (IT) industry and marketplace, and especially the embedded systems industry and marketplace, are changing very rapidly. End-user preferences and requirements change quickly, while also becoming increasingly sophisticated and interconnected with available IT environments and infrastructures. Time-to-market delays for an embedded system product can have a significant impact on revenue; a three-month delay releasing a product is usually critical.

In a survey of our customers, more than 60% of them indicated that *time to market* was their most important requirement. *Cost reduction* and the *integration of leading-edge technologies* were the second and third highest priorities, respectively. Because this means that customers want to draw upon design service offerings on demand, the service offerings should have a responsive, on-demand nature of their own. To realize an on-demand advantage for our design service offerings, we have prepared a series of platforms consisting of software stacks on various central processing units (CPUs) of varying performance. These platforms are reused to develop required systems. To reduce time and cost, we make use of software engineering tools and methodologies that are more typically used to develop large enterprise software.

0018-8646/04/$5.00 © 2004 IBM

Application-specific integrated circuit (ASIC) development, much like a complete embedded system, requires a comprehensive design approach. ASICs have become large enough to have various components—intellectual property (IP) macros—inside them. Like a printed circuit board design, such components are provided as libraries and reused. System-on-chip (SoC) designs—ASICs that have a CPU surrounded by various components—provide the major functions of embedded systems within the ASIC. Sets of tools that span the entire development process for ASICs are emerging. They provide capabilities for functional modeling, system architecture design using a high-level language, and logic and physical design in parallel with the development of the software to run on the ASIC. The design methodologies and tools we use to realize on-demand design services are discussed in the next section.

Assets at other levels, such as IP macros, device drivers, and middleware, are key factors in making technology and design services more beneficial. We also devote a section to discussing the technical aspects of two key assets.

On-demand design methodologies and tools

System concept
System prototyping on field-programmable gate arrays (FPGAs) has been adopted to validate systems. This prototyping can drastically reduce the time required for software-based logic simulation of an ASIC, and it can expand test coverage by working as a hardware accelerator [1]. Various methods such as multi-FPGA partitioning [2] have been proposed to enhance this capability. When prototyping a system on FPGAs, interfaces and partial behaviors of external devices can be modeled precisely, and sets of stimuli generated from the models are fed to logic simulations running on the FPGAs. The same logic design that will be used on a target SoC is used in the FPGAs to achieve complete logic validation. Even though the verification speed is still much faster than software-based logic simulation, real-time features are usually sacrificed, since the execution speed of FPGAs is typically five times slower than that of the target SoC when both integrated circuits are developed in the same technology.

Let us discuss the characteristics of SoC development for leading-edge digital consumer products and how this powerful method of FPGA-based prototyping could be improved.

Real-time system prototyping
One prototyping issue for digital consumer products is the lack of external device models. To meet customers' first priority, time to market, it is critical to reduce SoC development time as much as possible. However, it is very difficult to develop all of the external device models in a short time, and the specifications of the external devices are sometimes unclear or incorrect. A simple way to address this problem is to attach real external devices to the FPGAs and use the same logic code for them [3]. In particular, real applications could be executed to expand the test coverage. Since it is almost always troublesome to reduce the execution speed of external devices, it is desirable that the application run in real time. There have been rapid increases in speed and density for FPGAs [4], but it is still difficult to run all of the functions in FPGAs in real time for a target SoC, which can have up to five million gates.

Consequently, it is necessary to carefully reduce the execution speed of the internal logic components in the FPGAs, while the external interface continues to run at real-time speed. In addition, FPGAs with embedded CPUs, which have recently become available, enable us to easily emulate CPUs and surrounding logic components that are directly connected to the fast and wide SoC internal buses. Although it is beneficial to use these FPGAs, they may be structurally different from the target SoC; for example, the bus widths may be different, and a secondary processor bus may be lacking on the FPGAs. This means that it may no longer be possible to map the identical logic code for the target SoC onto the FPGAs in order to ensure logical equivalence between them at a cycle-time-accurate level. Instead, it is necessary to carefully investigate the characteristics of the target logic circuits (for example, determining how to synchronize logic components and determining the structural differences between the target SoC and the FPGAs) and then define strategies to validate the target logic, including logic mapping and test coverage.

Optimized development flow and environment
The FPGA-based emulation boards on which system prototypes are built are delivered to as many hardware and software developers as possible in order to reduce development time. These boards are usually expensive, since, by their nature, they include redundant functionality that could be used for various applications. An intelligent development flow with the most effective and timely verification tools for each step of the flow should be provided. **Figure 1** shows our development flow using system prototyping on FPGAs. Logic design and verification are roughly divided into three phases:

1. *Hardware acceleration for the logic simulation of components:* As soon as the system architecture is designed, logic design for each component is initiated. These logic components are verified by software-based logic simulation or by emulation on FPGAs. At this time, an FPGA board, which may have been developed during the previous development work or which may be

commercially available, is used, since it can work as a conventional hardware accelerator. The devices and components attached to a target logic component are partially modeled, and a limited number of stimuli are applied. The behavior of the CPU is provided by an instruction set simulator running on the host personal computer (PC) attached to the FPGA board. In parallel with the logic component development, an FPGA emulation board with real external devices can be designed and developed before the system integration test is started. This means that the FPGA emulation board should be developed within a couple of months from the beginning of the logic design. Embedded system software must also be developed for the test.

2. *Real-time system prototyping on FPGAs:* The logic components are connected to one another for the system integration test and mapped onto the FPGA emulation board, which works as an in-circuit emulator. Comprehensive system behavior, including the functions of all logic components in the target SoC and the related embedded software, can be verified by running real-time applications and validated before the physical design is started. Additional effort is required to complete the embedded software during the physical design and fabrication of the SoC chip.
3. *Real device verification:* When an engineering sample of the target SoC becomes available, a target card—on which the target SoC and external devices will be mounted—is developed. The final verification is executed with the real devices. However, if the system has been validated in the in-circuit emulation phase, no problems should be found at this stage. Therefore, this development flow can ensure first-time-right SoC development.

Software-based logic simulation, hardware acceleration of the simulation, and real-time FPGA system prototyping have complementary characteristics in terms of verification speed, signal monitoring capabilities, design equivalence, and compile time, as shown in **Table 1**. Because of the priority of execution speed, real-time system prototyping is used for the system integration test. If a problem is found, however, most developers prefer switching their verification strategy to use software simulation to investigate the behavior of the logic components, since this allows precise, flexible, and detailed signal monitoring. When a problem is identified, the developer then uses the hardware accelerator for rapid validation with much wider coverage. Therefore, it is important to allow the developer to choose appropriate methods and to switch seamlessly from one to another, especially in the system integration test.

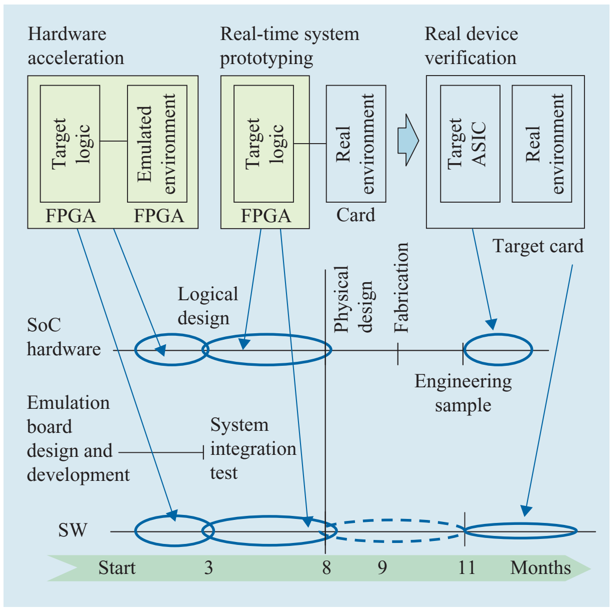

Figure 1

The three phases of our development flow using system prototyping on FPGAs. (The development period shows the case described in the implementation section of the text.)

Implementation

We developed an FPGA-based real-time system prototyping tool and used it for SoC development for a leading-edge digital consumer product. **Table 2** shows an overview of the target SoC. It consists of an IBM PowerPC* processor, image processing units, and various input/output (I/O) peripherals. In addition to the embedded CPU, the SoC is designed to interface with other intelligent components; thus, the behavior of the entire system is very complicated. To emulate the SoC, we developed an FPGA emulation board that consisted of three FPGAs, a Xilinx Virtex-II Pro** P20 with an embedded PowerPC 405 and 1.5M gates, a Vertex-II 6000 with 6M gates, and a Vertex-II 3000 with 3M gates, as well as external devices and interfaces with the other intelligent components. We ensured that the logic components in the target SoC were synchronized to one another using "handshakes," and that the external device interface could be handled, even when the internal clock speed was to some extent reduced. The speed of the processor local bus was reduced to one quarter of the real bus speed, and that of the output peripheral bus was reduced to one half (36.8 MHz). Of course, when clock speed is reduced by a different amount for a different set of logic components, some logic paths and states may not be verified. For example, an input buffer may become empty when the logic components of the previous stage

Table 1 Verification strategies.

Type of verification	*Characteristics*			
	Verification speed	*Signal monitoring*	*Design equivalency*	*Compile time*
Software-based logic simulation	Slow	Good	Good	Short
Hardware acceleration of the simulation	Fast	Good	Good	Fair
Real-time FPGA system prototyping	Very fast	Fair	Good (some limitations)	Long

Table 2 Target SoC overview.

CPU components	IBM PowerPC 440 coprocessor, image processing units, memory interface, custom I/O buses, standard serial I/Os.
Gate count	5M gates
Clock speed	442 MHz (CPU) 147 MHz (internal bus)
Technology	0.13 μm

run more slowly. We compensated for this phenomenon by adjusting the bus arbitration priority. In a conventional method, the target logic is designed to be suitable for the performance of the target SoC and it is then converted to the FPGAs. In contrast, we took its ability to run on the FPGAs into account at the outset.

We developed the FPGA emulation board in three months, finishing it just before the system integration test was scheduled to begin. During the test, we were verifying the target logic on the real-time system prototype by running real application software. While doing so, we found a few critical problems in the external interface. These problems were caused by unclear and incorrect specifications of the external devices and would not have been detected by conventional system prototyping methods. Finally, eight months after the SoC development began, the complete logic circuitry of the target SoC was validated.

Examples of common IP assets for embedded systems

As discussed in the section above, one of the important aspects required to satisfy customer requirements and to strengthen the competitiveness of the design service business in the embedded devices area is to have strong IP macros that are commonly used for most applications. In general, embedded systems consist of a wide range of technologies and components, depending on their functions; hence, it is quite difficult to develop some IP assets before the requirements are fixed. However, certain important technologies and components, such as security and communication technologies, are shared in common for most systems. We are working in these technical areas to develop strong design service offerings. Tailor-made IP macros can be quickly plugged into SoCs or ASICs and can provide quick, flexible, precise, and advanced characteristics for design service offerings.

Security architectures and the resulting cryptographic hardware and software are essential requirements for most embedded systems. However, since cryptographic algorithms are typically based on deep mathematics, their implementation in circuits results in large numbers of transistors. This can greatly degrade performance, thus creating a challenge to provide security in embedded systems. In some embedded applications, the performance of cryptographic circuits is critical, and should not be slower than the external interface speed, such as a 10-Gb/s optical communication link. However, most embedded applications must accommodate small or battery-operated devices, which generally makes the requirements for low power and small area more important than performance. Thus, an IP macro that delivers security features should be flexible in terms of performance, power dissipation, and area size, and should have a scalable design to meet diverse application requirements. Our circuit optimization techniques for cryptographic circuits are discussed in the section on optimized cryptographic circuits.

Communications technology is another example of an application typical of embedded systems. Advances in wireless technologies, such as IEEE 802.11a/b/g** and Bluetooth**, allow people to be connected at any time and in any place. However, before devices are connected, a user must set up network parameters (such as a service set identifier, the use of a Dynamic Host Configuration Protocol, and an Internet gateway) each time the location changes. Inappropriate settings can result in both a frustrating experience and a security exposure. Setting the network parameters is not easy for typical end users. For these embedded system users, it may be difficult or effectively impossible. Our unique autonomic network configuration mechanism for setting up network parameters frees users from such nuisances, as described in the section on autonomic network configuration technology which follows.

Optimized cryptographic circuits

Why cryptographic hardware?

Cryptographic algorithms require heavy mathematical computation; thus, until recently, software implementations were very slow and could sometimes greatly decrease system performance. Today, however, 32-bit microprocessors are fast enough to perform cryptographic operations for uses such as Internet shopping, online trading, and secure e-mail. On the other hand, 8-bit and 16-bit microcontrollers for embedded use are still not fast enough for these functions; it may take them several tens of seconds to several minutes to perform public-key cryptographic operations. Also, in high-end server systems, it is hard to obtain adequate performance using cryptographic software, because thousands of secure transactions may be entered at one time. With the rapid expansion of broadband communication services, much higher performance will be required for these servers. Considering cost, if the main use of those servers is cryptography, well-designed hardware is hundreds of times better than general-purpose CPUs. In addition, cryptographic hardware provides a higher security level. For these reasons, the demand for high-performance cryptographic hardware is growing.

Since IBM developed its first cryptographic chip in 1997 [5], we have introduced many prototypes and products: a fingerprint identification unit [6], a secure socket layer (SSL) accelerator board, a cryptographic hardware IP evaluation board, a security accelerator for the PowerPC processor, and a secure hard disk drive.

Cryptographic algorithms are categorized into *symmetric key cryptography,* used for data encryption, *public key cryptography,* used for digital signatures, and *secret key agreements*. RSA[1] [7] is the *de facto* standard for public-key cryptography. Our first cryptographic chip with 0.5-μm complementary metal oxide semiconductor (CMOS) technology achieved the best performance in terms of speed, size, and power consumption in 1997 [5].

Because of the high-speed and low-power features of our RSA chip, it was used for the fingerprint identification unit. Many such products use an image sensor unit to scan the fingerprint, which is then sent to a PC or server system for pattern-matching operations. However, such systems store the highly personal fingerprint data on a hard disk drive or let a third person administer the data. Therefore, privacy issues arise; hence, the fingerprint identification unit should be tamper-resistant. Because the fingerprint identification unit itself stores the data and does everything—pattern matching, data encryption, digital signature generation, and key generation—it is extremely tamper-resistant. The unit also supports a public-key infrastructure interface by using our RSA chip, enabling it to achieve a very high level of security, while user privacy is protected.

In addition to the RSA chip, we developed IP macros for almost all of the major cryptographic algorithms. They are coded in a pure Very high-speed integrated circuit Hardware Description Language (VHDL) and thus are easy to map on both IBM and vender CMOS technologies. Their functions were verified on FPGA boards. Elliptic curve cryptography (ECC) is expected to become the public-key cryptographic standard for the next generation, because its operands are very short and its operations are very fast compared with RSA. However, ECC requires deep mathematical knowledge to develop the hardware and to generate secure system parameters, so we are now doing research in this area of mathematics. As a result, we have developed very compact, flexible, and fast elliptic curve cryptographic hardware and we have merged the RSA macro with it.

We have also developed many symmetric-key cryptographic circuits. Our Data Encryption Standard (DES) [8] macro was integrated into the PowerPC 405LP [9]. The Advanced Encryption Standard (AES) [10] was standardized as a Federal Information Processing Standard in the year 2001 to replace DES. We then developed a high-performance AES circuit and achieved the smallest size and highest throughput in the world by applying both circuit design and mathematical techniques [11].

Table 3 shows our various cryptographic hardware IP macros. The macros were implemented on FPGA boards to demonstrate their functionality and performance. The design parameters, such as clock cycles and gate counts, can be chosen flexibly to achieve target performance and optimal circuits. In the following sections, we explain how to achieve high-performance scalable circuits with a block cipher AES and public-key ciphers with ECC [12, 13].

Advanced Encryption Standard

Composite field S-Box

Figure 2 shows an Advanced Encryption Standard (AES) encryption process under a 128-bit secret key. Eleven sets of round keys are generated from the secret key and fed to each round of the ciphering block. The round operation is a combination of four primitive functions: `SubBytes` (sixteen 8-bit S-Boxes), `ShiftRows` (byte boundary rotations), `MixColumns` (4-byte $\times$ 4-byte matrix

[1] The letters stand for the names of the inventors: Rivest, Shamir, and Adleman.

Table 3 IBM cryptographic hardware IP macros.

Category	*Algorithm*	*Speed*	*Gate counts* (Kgates)	*Notes*
128-bit block cipher	AES	311 Mb/s–3.5 Gb/s	6–37	S-Box architecture
		8.9 Mb/s–11.3 Gb/s	100–450	T-Box architecture
	Camellia	328 Mb/s–2.1 Gb/s	7–30	NESSIE recommendation
64-bit block cipher	DES Triple-DES	1.0–3.2 Gb/s 334 Mb/s–1.1 Gb/s	6–17	*De facto* standard
	KASUMI	287 Mb/s–1.6 Gb/s	5–8	Standard cipher for third-generation cellular phone
Stream cipher	RC4	1.1 Gb/s	35	WEP and Lotus Notes* support
Hash function	MD5	775 Mb/s	25	Single core supports two functions
	SHA-1	600 Mb/s		
	SHA-1 SHA-256/-384/-512	600 Mb/s–1.4 Gb/s 700 Mb/s–2.6 Gb/s	26–34	Single core supports four functions
Public-key cipher	RSA	21–1,003 operation/s (for 1,024 bits)	20–107 + 2 KB SRAM	Dual-field multiplier supports GF(p) and GF(2^n) operations. No limitations for operator size, field, or curve parameters.
	ECC on GF(p) ECC on GF(2^n)	67–413 EC-DSA/s 575–2,632 EC-DSA/s		

operation), and `AddRoundKeys` (bit-wise XOR). In decryption, the inverse functions (`InvSubBytes`, `InvShiftRows`, and `InvMixColumns`, with `AddRoundKey` as its own inverse) are executed in reverse order. The key scheduler uses four S-Boxes and 4-byte constant values $Rcon(i)$ ($1 \le i \le 10$). In decryption, these sets of keys are used in reverse order.

The most critical hardware components are the S-Boxes, and they sometimes consume half of the hardware resources and computation time. Therefore, we optimized them to achieve compact, high-speed AES hardware designs. The AES S-Boxes consist of multiplicative inversion on a Galois field GF(2^8) and affine transformations defined as 8×8 XOR matrices. However, most of the conventional designs have not used the arithmetic structure, and S-Box circuits have been automatically synthesized from lookup tables that define the relationship between input and output data. We first introduced the composite field inverter on GF($((2^2)^2)^2$) shown in **Figure 3** for the compact S-Box [11]. The inverter has a nested structure, and the arithmetic logic is highly optimized at each nesting level. For the field conversion between GF(2^8) and GF($((2^2)^2)^2$), the isomorphism functions δ and δ^{-1} are used. These functions are defined as 8×8 XOR matrices, so we merged each of them with the affine transformation into one XOR matrix. The composite field inverter is shared between encryption and decryption S-Boxes (`SubBytes` and `InvSubBytes`). As a result, the gate counts for the S-Boxes were reduced to at least one third in comparison with the lookup-table S-Boxes.

Scalable AES hardware architecture

Figure 4 shows our compact AES architecture with the data I/O buses omitted for simplicity. The 128-bit data is divided into four 32-bit columns, and each column is processed by a 32-bit ciphering block. In addition to S-Box optimization, `MixColumns` and `InvMixColumns` are merged and compressed by factoring common terms. The four S-Boxes in the ciphering blocks are reused by the key scheduler. This architecture takes five cycles for one round, and a total of 54 cycles for one encryption or decryption.

We also designed various circuits by changing the number of primitive components. Their performances, using a 0.13-μm CMOS standard cell library, are shown in **Figure 5** along with those for other recent AES designs [14–18]. All of our designs show much better performance compared with the others. In addition, our architectures show good scalability between throughputs as a function of gate count. We also developed a logic synthesis methodology called *twisted-BDD* for high-speed combinatorial circuits and obtained an ultrahigh-speed AES design whose throughput is higher than 10 Gb/s [19].

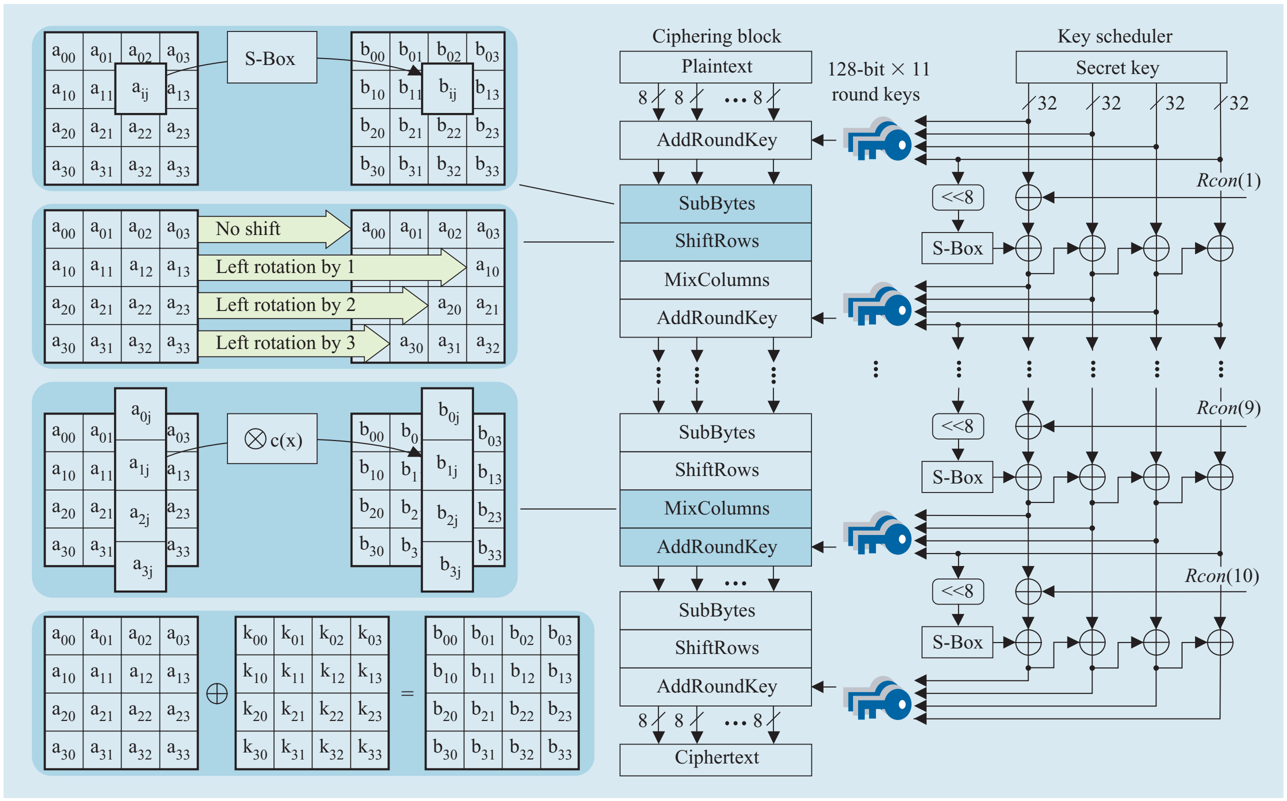

Figure 2

AES encryption process under a 128-bit secret key.

Elliptic curve cryptography

Dual-field multiplier

Elliptic curves are classified roughly into two groups: those defined over prime fields GF(p) and those defined over binary fields GF(2^n). Many ECC hardware architectures have been proposed [20–32], but most of them have little flexibility, since the parameters for the curves or the fields were predetermined and could not be changed. The Montgomery multiplication algorithm [33, 34] was proposed for speeding up modular multiplication on GF(p), which is the most computationally demanding operation in the ECC hardware, and it was later extended to GF(2^n) [35]. The algorithm can be applied to arbitrary elliptic curves on GF(p) and also on GF(2^n) defined as polynomial fields.

From the point of view of the hardware, a multiplier-based architecture is suitable for a compact, high-speed Montgomery multiplier. An r-bit × r-bit multiplier that can directly interface with an r-bit system bus is the best choice, but some conventional designs choose an n-bit × r-bit multiplier or an n-bit adder, where n is the field size. Then the arithmetic unit must wait until all n bits of the

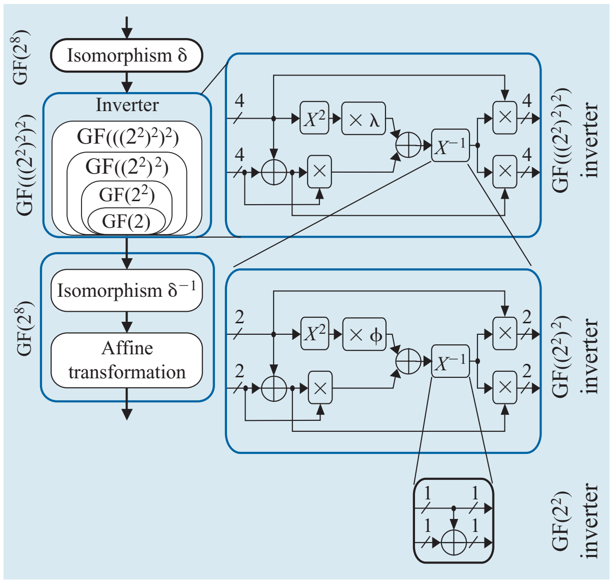

Figure 3

Composite field inverter on GF($((2^2)^2)^2$) for the compact S-Box.

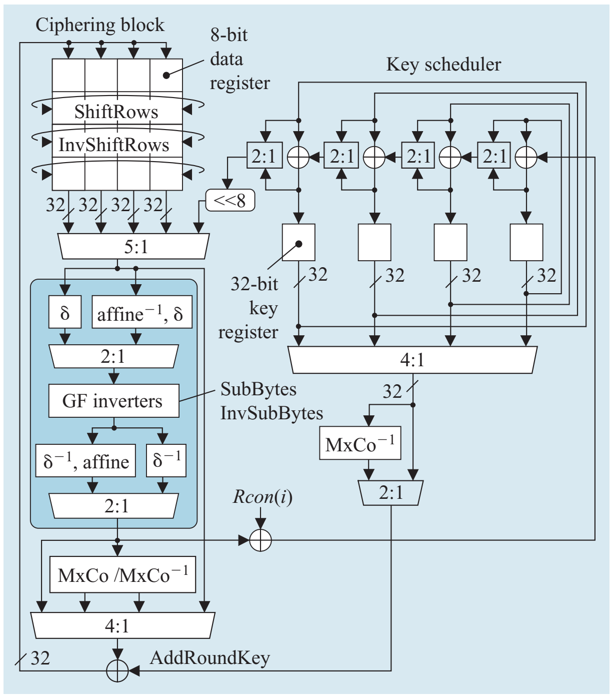

Figure 4

Compact AES architecture (the data I/O buses are omitted for simplicity).

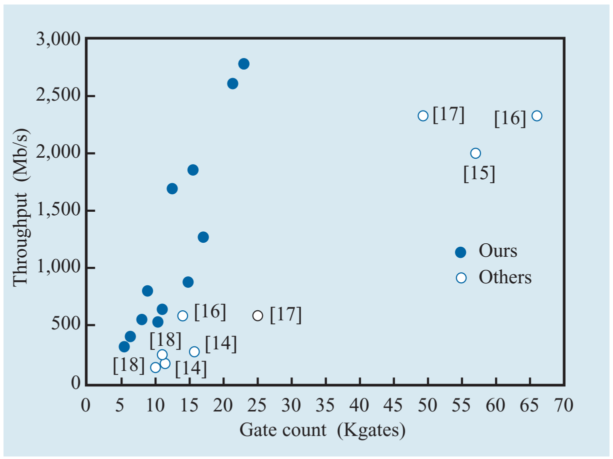

Figure 5

Performance of various circuits we designed by changing the number of primitive components, using a 0.13-μm CMOS standard cell library.

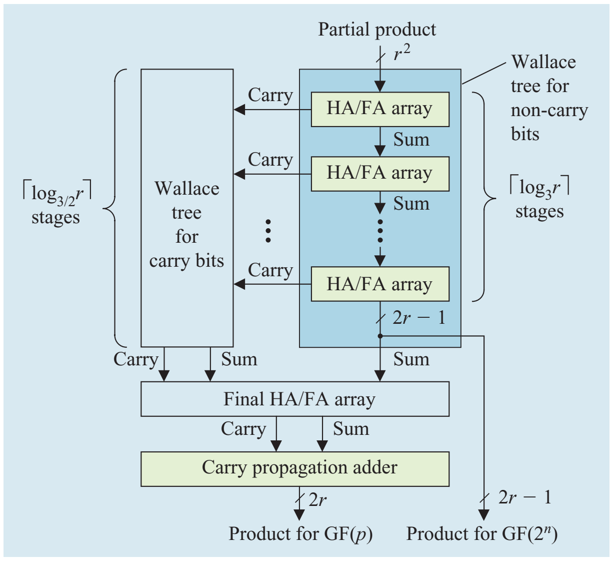

Figure 6

Basic structure of our dual-field multiplier.

data have arrived via the r-bit bus. If a special n-bit-wide bus is used, it diminishes the flexibility of the operator size.

Therefore, we developed a Montgomery multiplier using an r-bit $\times$ r-bit multiplier that supports the two finite fields, GF(p) and GF(2^n) [36]. **Figure 6** shows the basic structure of our dual-field multiplier. The parallel multiplier contains a Wallace tree[2] part to sum up the r^2-bit partial product in carry-save (redundant binary) form, and a carry-propagation adder to convert the redundant binary number into a normal binary number. In our multiplier, the GF(2^n) operation is successfully integrated into the Wallace tree block without additional hardware.

Scalable ECC processor

Figure 7 shows the block diagram of our ECC processor using the dual-field multiplier. The sequencer block has a four-level hierarchical structure and supports a variety of arithmetic functions in addition to those required for ECC. Level 1 supports low-level functions, such as basic modular arithmetic and Montgomery operations. Level 2 executes modular exponentiation, multiplicative inversion, and pre- and post-processing for the Montgomery operations by using Level 1 functions. The sequencer for GF(p) at Level 2 provides prime number generation by using Fermat's primality test, RSA key generation by the extended Euclidean algorithm, and the Chinese remainder theorem operation. Level 3 supports basic elliptic curve (EC) operations and on-the-fly redundant binary conversion. The highest level, Level 4, executes the digital

[2] The Wallace tree receives partial products of two numbers and feeds an intermediate result in carry-save format to a carry-propagation adder. The tree comprises a number of half and full adders in an irregular structure.

signature algorithm (DSA) and EC-DSA signature generation and verification [37]. Our architecture has a clearly separated control structure; thus, it is easy to design and modify the logic, and it has high flexibility for functional extensions.

The ECC processors were synthesized for 8–64-bit multipliers by using the 0.13-μm CMOS standard cell library. **Figure 8** shows the performance of each implementation. The number of EC scalar multiplications per second for 160–256-bit curves is shown as a function of the number of gates. As shown in the figure, our hardware architecture provides good scalability in terms of speed, area, and operator size.

Table 4 shows performance comparisons with conventional hardware for EC scalar multiplications in GF(2^n) and GF(p). Our EC processor shows the fastest operation times in both fields, even though most of the conventional circuits support a single field and use special fields or EC parameters to boost speed and to reduce hardware resources.

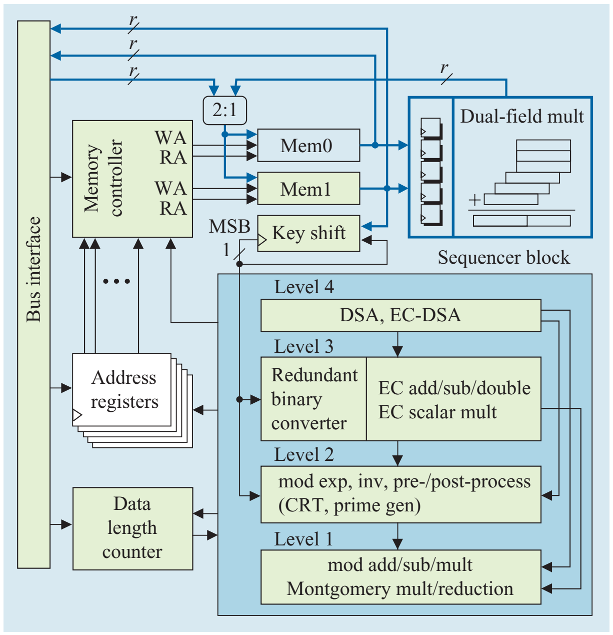

Figure 7

Block diagram of scalable ECC processor using the dual-field multiplier.

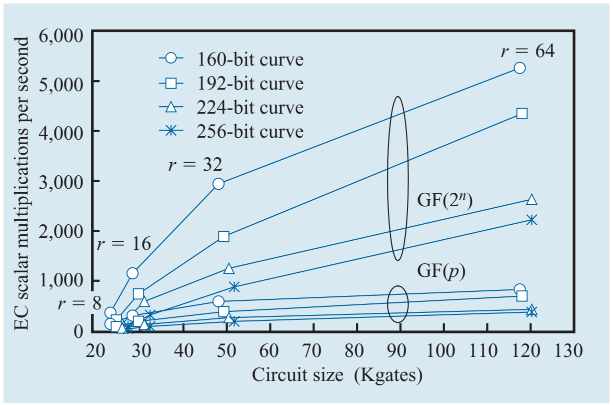

Figure 8

Tradeoff between speed and gate counts for EC scalar multiplication.

Autonomic network configuration technology

Requirements around networking

Embedded systems are often small and have poor user interfaces. They can be tiny sensors or controllers without a large display or a keyboard. Even so, embedded systems are increasingly capable of accessing networks in order to exchange data and control information with peer embedded systems or with management server systems. However, they share common challenges associated with network configuration:

1. Quite a few network and security parameters must be appropriately configured before the system can be connected to the network.
2. These parameters must be modified each time the system is connected to a different network.
3. Network problems, if present, are usually complicated and difficult to solve.

The complexity of mobile environments can occasionally annoy its users and make network and security configuration difficult. When users encounter network problems, they can become frustrated; it can be difficult for them to locate the cause of their network troubles. When that happens, many users know of no other course of action than to wait and hope their network recovers by itself.

Autonomic network configuration

To address this difficult situation, we introduced a holistic approach to autonomically solve a wide range of network problems without users being aware. We proposed an autonomic network configuration (ANC) technology that takes an autonomic client approach to mask complicated network problems. It cycles repeatedly through four phases: a monitor, an analyzer, a planner, and an executor. It realizes the four pillars of autonomic computing attributes: self-configuring, self-optimizing, self-healing, and self-protecting.

Table 4 ECC hardware performance comparisons.

Reference	*Field*	*Platform*	*Maximum frequency* (MHz)	*EC mult time* (ms)	*Notes*	
EC scalar multiplications in GF(2^n)						
Ours	GF(2^{160})	0.13-μm CMOS ASIC	510.2	0.19	64-bit multiplier	
[20]	GF(2^{155}) ONB	ASIC	40	3.9	Massey–Omura multiplier	
[21, 22]	GF(2^{155}) ONB	Xilinx xc4020XL	15	15.9*	Massey–Omura multiplier	
[23]	GF(2^{155}) ONB	Xilinx xcv300-4	36	6.8	Massey–Omura multiplier	
[24]	GF(2^{155}) ONB	Xilinx xc4085XLA	37	1.29	Massey–Omura multiplier	
[25]	GF(2^{53}) ONB	Xilinx xc4044XL		2.4	Massey–Omura multiplier	
[26]	GF($((2^2)^4)^{21}$)	Xilinx xc4062	16	4.5	Special composite field multiplier	
[27]	GF(2^{167})	Xilinx xcv400E	76.7	0.21	167-bit $\times$ 16-bit multiplier and 167-bit $\times$ 167-bit squarer For $P(x) = x^{167} + x^6 + 1$	
[28]	GF(2^{191})	Xilinx xc4000XL		17.71	Standard form	$P(x) = x^{191} + x^9 + 1$
				11.82	Hessian form	
[29]	GF(2^{163})	0.25-μm CMOS ASIC	66	1.1	Random curve	288-bit $\times$ 8-bit multiplier
				0.65	Koblitz curve	
		ALTERA EPF10K259AGC599-2	3	80.3	Random curve	82-bit $\times$ 4-bit multiplier
				45.6	Koblitz curve	
[30]	GF(2^{160})	ASIC	50	7*	1024-bit adder	
EC scalar multiplications in GF(p)						
Ours	GF(p) 192 bits	0.13-μm CMOS ASIC	137.7	1.44	64-bit multiplier	
[31]	GF($2^{192} - 2^{64} - 1$)	Xilinx xcv1000E-8	40	3	192-bit $\times$ 8-bit multiplier	
[32]	GF($2^{192} - 2^{64} - 1$)	ASIC	50	30	1 unit	ARM7 + arithmetic unit Hessian form with $a = p - 3$

*Authors' estimate based on reference.

The autonomic client approach is more effective in solving complicated network problems than a collection of straightforward approaches to each problem. The first reason for this is that a problem may be caused by one critical element, or it may result from an interaction of noncritical elements. For example, a fatal error with the default gateway stops everything, while unlucky coincidences with simultaneous bursts of traffic on two intermediate routers in the Internet can also stop Internet accesses. Many network components affect network performance, and they change their status and interact with each other continuously. The second reason for the effectiveness of the ANC technology is that there can be more than one good solution. Users are happy as long as they can enjoy network services with reasonable performance. They are willing to accept any good solution, whether or not it is the best solution. The third reason is that users may have several implicit and explicit levels of network performance requirements, depending on location, time, content, and cost.

Working around the problem is also a reasonable solution that users can sometimes achieve by themselves. Since they are not the network administrator, their goal is to do something to access network services (such as the location server), but not to address the underlying network problems. If they have alternatives, they may simply choose to bypass problems instead of removing them. For instance, switching from a wireless local area network to an Ethernet connection can be a workaround for troubles with a hub, and switching to another proxy server will bypass a faulty proxy server.

Figure 9 shows the ANC system structure. The four ANC entities—monitor, analyzer, planner, and executor—correspond to the four-phased autonomic approach. They communicate with one another in order to reduce the frequency of interactions and to maximize the effectiveness of ANC features. Note that each entity is ready for customization to accommodate a wide range of system requirements.

Monitor

The monitor detects any unusual states (e.g., increases in error rates, drop rates, and collision rates) by constantly monitoring network activities at various levels of the Transmission Control Protocol/Internet Protocol (TCP/IP) architecture hierarchy. For example, it looks at Ethernet headers in the network layer, IP headers in the Internet layer, TCP headers in the transport layer, and HyperText Transfer Protocol (HTTP) headers in the application layer. The monitor can report on fatal errors (such as drops and bit errors) and many sensitive symptoms (such as increasing delay).

ANC has two types of engines: a passive monitor and an active monitor. The passive monitor is completely silent; it simply tracks the packets initiated by applications or the operating system (OS). It can monitor the normal network traffic, detecting many kinds of errors and unusual behavior. The active monitor, on the other hand, is a kind of explorer. It creates its own network packets to check the state of network components. This monitor can be used to produce network packets for diagnostics when the passive monitor detects evidence of certain kinds of problems that produce only indirect symptoms, such as delays in network transmissions. It is also necessary in order to probe alternate paths such as another network interface card (NIC) or gateway, because they may otherwise not be activated or used by the OS.

The passive and active monitors complement each other. For example, they can check the availability of the primary domain name system (DNS) [38] server in the following way. The passive monitor constantly watches for DNS query and response packets resulting from the normal operations of the user, while the active monitor periodically sends its own DNS query packets to the server to check its availability or responsiveness, as may be necessary when there are no application-driven DNS query packets for a long time.

Analyzer

The function of the analyzer is to determine what is happening in the network by analyzing the results reported by the monitor. It directs the monitor to the target (such as DNS packets to the primary DNS server) and the method (passive or active, and if active, how often). It changes its direction in accordance with the results reported by the monitor. As an example of its operation, assume that the analyzer has the assignment of reporting to the planner whenever the DNS server becomes unavailable. It asks the passive monitor to constantly track DNS packets and it also asks the active monitor to send a dummy DNS packet when there have been no DNS packets for a sufficiently long time.

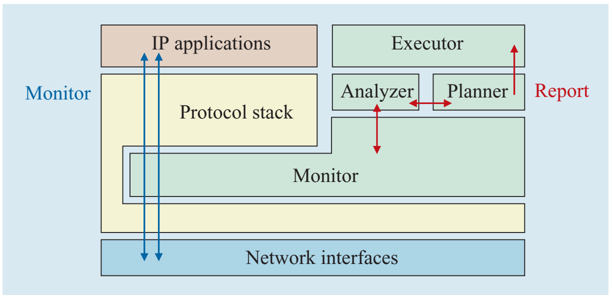

Figure 9

Structure of autonomic network configuration (ANC) system.

Planner

The planner maintains the configuration policies in terms of network performance, availability, and costs. It communicates with the analyzer to obtain the necessary information to support the given policies. To illustrate its operation, assume that the planner has a policy goal of keeping the network available, and it develops one or more recovery plans when the network becomes unavailable. It may ask the analyzer to check the availability of the DNS server and default gateway. If it detects any trouble with the primary DNS server, it will check the availability of the secondary DNS server. If there is more than one NIC, it can configure the analyzer to check an alternative NIC to determine whether a backup network connection is available.

Executor

Once the executor receives reports from the planner about the problems and alternate recovery plans, it applies the configuration policy from the planner and makes the final decisions on changing the configuration. It may receive more than one policy from the planner and may even receive some contradictory recovery plans. It is the responsibility of the executor to resolve any contradictions.

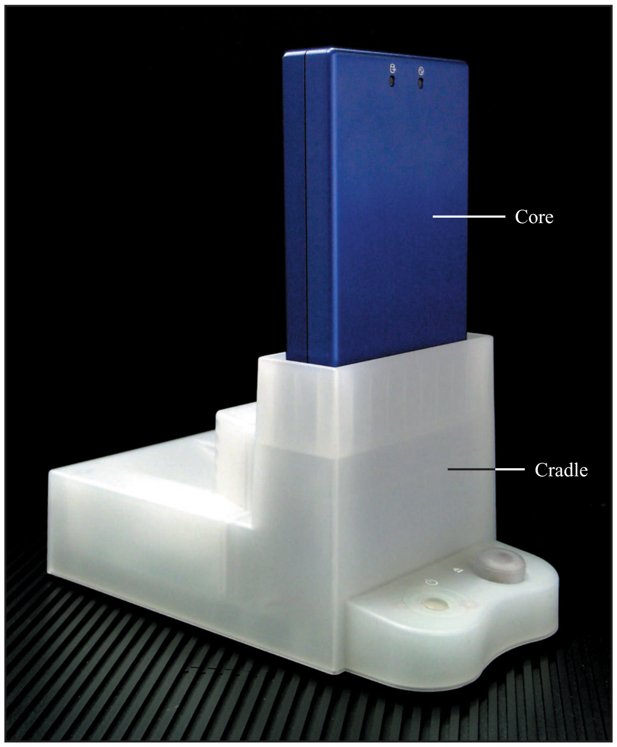

Figure 10

The IBM mobile computer core contains a CPU and disk drive. The user can carry it around and make use of any display and keyboard equipped with a cradle.

Network environment detection service for an IBM PC core system

The IBM PC core system[3] consists of a core and cradle, as shown in **Figure 10**. The core is portable and contains a CPU, memory, and disk drive. The cradle has external connectors to the power, display, keyboard, mouse, and network. Thus, users can carry their cores with them and, as long as there is an available cradle connected to a display and keyboard, they can plug in their core and work in their familiar computing environment. This system was developed in the IBM Yamato Laboratory on the basis of the modular computer design concept.

The core system is ideal for an organization in which people share computers installed in many locations, such as large enterprises and universities. There can be several cradles in different locations, each set up with a display, keyboard, mouse, and network. Anyone can use a personalized core at any of these locations. Their computing environment would remain the same, just as if they were transporting their entire computer system. At the same time, cradles can be shared with others. The cradle enables any core to recover its computing environment.

However, most of the time users must do more than simply dock their cores to the cradle. Although the OS in the core can detect changes in the hardware configuration, it does not automatically configure network parameter settings. Users usually have to set up the IP parameters, proxy server, and file sharing each time they move from one cradle to another, but changes in network environments can be so frequent that users are often frustrated.

ANC technology can be used to detect the network environment and supply the core system with a capability for autonomic network configuration. ANC monitors all incoming network packets and analyzes them to identify the media access control (MAC) addresses of key network components, such as the default gateways, the DNS servers, and the Dynamic Host Configuration Protocol servers. This eliminates network setup frustration and makes the core system more attractive, even to users unfamiliar with the network.

Location-based systems with the IBM PC core system

We demonstrated ANC capability for location-based network services using the core system. In the demonstration, there are two locations, an office and a warehouse, and two applications, mail and inventory. The mail application launches when the core connects to the office environment, and the inventory application launches when it connects to the warehouse environment.

In this system, ANC controls not only the settings, but also the application launches by detecting the network environment to which the core is connected. When the ANC is equipped with network environment detection, it gives the core location-awareness features and can support busy mobile users actively moving from one location to another.

Core technology ready for customization

ANC is an important core technology for embedded systems to enable network communication because it autonomically maintains network connectivity. Although the ANC architecture is independent of the system and application, and the ANC monitor can track any network packets, we did not develop a general network configuration tool. Such a tool usually has too many functions and a complicated user interface for setting parameters, and it may still not meet user requirements completely. Instead, we developed ANC technology as a core technology that can easily be customized. Offerings such as innovative IP macros are key to an on-demand design service. Because ANC is a core technology that can easily be tailored in accordance with functional and system

[3] The IBM PC core system was demonstrated at the IBM Forum 2004, held in February 2004 in Tokyo, Japan. It is designed to deliver corporate solutions with a new compact unit.

requirements, it can be delivered in a small module with a short development time.

The location-based system for embedded systems is an example of the type of applications that ANC can make possible with its network environment detection service. ANC directly monitors low-level network packets and quickly and safely configures the computing environment for a specific location, even before the IP address is allocated.

In addition to the network environment detection service, ANC technology can support other services, such as security and performance management services. We have core technologies and implementations ready for customization to provide reasonable solutions for real requirements and problems with a very short development time.

Concluding remarks

In this paper, we have discussed fundamental enablers for the on-demand infrastructure for design services and specific enablers for the technology services business. Three technical topics have been discussed: design methodologies and tools for rapidly developing embedded system hardware and software, cryptographic hardware IP macros, and an ANC core.

In terms of design methodologies and tools, we elaborated on the characteristics of SoC development for leading-edge digital consumer products and on the requirements for system prototyping on FPGAs. We explained that because of the lack of external device models, a real-time interface to external devices is desired. On the basis of this observation, we described our development of the world's first FPGA-based, real-time system prototyping tool for PowerPC SoC, which works in a real environment. We defined the design flow and described its application to the development of a leading-edge digital consumer product. We confirmed that the tool and methodology fit well into SoC development in this domain.

For device security, we discussed the cryptography IP macros for ASIC development. We showed that these cryptography IP macros offered world-class advantages in their optimization level for performance, silicon efficiency, and power dissipation.

Finally, we discussed ANC technology, which can autonomically maintain network connectivity, rendering it an important core technology for embedded systems dependent upon network communication.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of Sun Microsystems, Inc., Xilinx, Inc., IEEE, Bluetooth Special Interest Group, or Intel Corporation in the United States, other countries, or both.

References

1. S. Walkers, "Reprogrammable Hardware Emulation Automates System-Level ASIC Validation," *Proceedings of the Wescon/'90 Conference Records, Electron Conventions Mgt.*, November 1990, pp. 140–143.
2. W.-J. Fang and A. C.-H. Wu, "Performance-Driven Multi-FPGA Partitioning Using Functional Clustering and Replication," *Proceedings of the 35th Design Automation Conference (DAC)*, June 1998, pp. 283–286.
3. M. Meerwein, C. Baumgartner, T. Wieja, and W. Glauert, "Embedded Systems Verification with FPGA-Enhanced In-Circuit Emulator," *Proceedings of the 13th International Symposium on System Synthesis*, September 2000, pp. 143–148.
4. Virtex-II Pro Platform FPGAs: Complete Data Sheet, Xilinx, DS083 March 9, 2004; see *http://direct.xilinx.com/bvdocs/publications/ds083.pdf*.
5. A. Satoh, Y. Kobayashi, H. Niijima, N. Ohba, S. Munetoh, and S. Sone, "A High-Speed Small RSA Encryption LSI with Low Power Dissipation," *Proceedings of the 1997 Information Security Workshop (ISW '97)*, September 1997, pp. 174–187.
6. Sony Electronics Inc.; see *http://bssc.sel.sony.com/Professional/puppy/index.html*.
7. R. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Commun. ACM* **21,** No. 2, 120–126 (February 1978).
8. National Institute of Standards and Technology, "Data Encryption Standard (DES)," FIPS Publication 46-3, October 1999; see *http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf*.
9. "IBM PowerPC 405LP," IBM PowerPC processor news (October 2001); see *http://www-306.ibm.com/chips/products/powerpc/newsletter/oct2001/new-prod2.html*.
10. National Institute of Standards and Technology, "Advanced Encryption Standard (AES)," FIPS Publication 197, November 2001; see *http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf*.
11. A. Satoh, S. Morioka, S. Munetoh, and K. Takano, "A Compact Rijndael Hardware Architecture with S-Box Optimization," *Advances in Cryptology (ASIACRYPT 2001)*; published in *Lecture Notes in Computer Science* **2248** (C. Boyd, Ed.), 239–254 (2001).
12. V. S. Miller, "Use of Elliptic Curve in Cryptography," *Proceedings of Advances in Cryptology (Crypto '85)*; published in *Lecture Notes in Computer Science* **218** (H. C. Williams, Ed.), 417–426 (1986).
13. N. Koblitz, "Elliptic Curve Cryptosystems," *Math. Comp.* **48,** 203–209 (1987).
14. M. Aigner, "Scalable AES HW-Module," Institute for Applied Information Processing and Communications, Graz University of Technology, July 2001; see *http://www.iaik.tu-graz.ac.at/research/vlsi%20design/aes%20for%20smartcards/aes-brochure/IAIK_AES_Module.pdf*.
15. Helion Technology, Overview Datasheet—High Performance AES (Rijndael) cores for ASIC, 2002; see *http://www.heliontech.com/downloads/aes_asic_helioncore.pdf*.
16. Ocean Logic Pty Ltd., OL_AES-AES Core family Rev 1.4; see *http://www.ocean-logic.com/pub/OL_AES.pdf*.
17. Amphion Semiconductor, CS5265/75-AES Simplex Encryption/Decryption Cores; see *http://www.amphion.com/e-d.html*.
18. S. Mangard, M. Aigner, and S. Dominikus, "A Highly Regular and Scalable AES Hardware Architecture," *IEEE Trans. Computers* **52,** No. 4, 483–491 (April 2003).
19. S. Morioka and A. Satoh, "A 10 Gbps Full-AES Crypto Hardware with a Twisted-BDD S-Box Architecture," *Proceedings of the 2002 IEEE International Conference on*

Computer Design: VLSI in Computers and Processors (ICCD'02), September 2002, pp. 98–103.
20. G. B. Agnew, R. C. Mullin, and S. Vanstone, "An Implementation of Elliptic Curve Cryptosystems over F_2155, *IEEE J. Selected Areas in Commun.* **11,** No. 5, 804–813 (June 1993).
21. S. Sutikno, A. Surya, and R. Effendi, "An Implementation of ElGamal Elliptic Curves Cryptosystems," *Proceedings of the 1998 IEEE Asia-Pacific Conference on Circuits and Systems (APCCAS'98)*, November 1998, pp. 483–486.
22. S. Sutikno, R. Effendi, and A. Surya, "Design and Implementation of Arithmetic Processor F_2155, for Elliptic Curve Cryptosystems," *Proceedings of the 1998 IEEE Asia-Pacific Conference on Circuits and Systems (APCCAS'98)*, November 1998, pp. 647–650.
23. K. H. Leung, K. W. Ma, W. K. Wong, and P. H. W. Leong, "FPGA Implementation of a Microcoded Elliptic Curve Cryptographic Processor," *Proceedings of the 2000 IEEE Symposium on Field Programmable Custom Computing Machines (FCCM)*, April 2000, pp. 68–76.
24. M. Ernst, S. Klupsch, O. Hauck, and S. A. Huss, "Rapid Prototyping for Hardware Accelerated Elliptic Curve Public-Key Cryptosystems," *Proceedings of the 12th International Workshop on Rapid System Prototyping (RSP 2001)*, June 2001, pp. 24–31.
25. L. Gao, S. Shrivastava, and G. E. Sobelman, "Elliptic Curve Scalar Multiplier Design Using FPGAs," *Proceedings of the First International Workshop on Cryptographic Hardware and Embedded Systems (CHES'99)*; published in *Lecture Notes in Computer Science* **1717** (Ç. K. Koç and C. Paar, Eds.), 257–268 (August 1999).
26. M. C. Rosner, "Elliptic Curve Cryptosystems on Reconfigurable Hardware," Master's Thesis, ECE Department, Worcester Polytechnic Institute, Worcester, MA, May 1998.
27. G. Orlando and C. Paar, "A High-Performance Reconfigurable Elliptic Curve Processor for GF(2^m)," *Proceedings of the Second International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2000)*; published in *Lecture Notes in Computer Science* **1965,** 41–56 (August 2000).
28. N. P. Smart, "The Hessian Form of an Elliptic Curve," *Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2001)*; published in *Lecture Notes in Computer Science* **2162** (Ç. K. Koç, D. Naccache, and C. Paar, Eds.), 118–125 (May 2001).
29. S. Okada, N. Torii, K. Itoh, and M. Takenaka, "Implementation Coprocessor over GF(2^m) on an FPGA," *Proceedings of the Second International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2000)*; published in *Lecture Notes in Computer Science* **1965,** 25–40 (August 2000).
30. J. Goodman and A. Chandrakasan, "An Energy Efficient Reconfigurable Public-Key Cryptography Processor Architecture," *Proceedings of the Second International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2000)*; published in *Lecture Notes in Computer Science* **1965,** 175–190 (August 2000).
31. G. Orlando and C. Paar, "A Scalable GF(p) Elliptic Curve Processor Architecture for Programmable Hardware," *Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2001)*; published in *Lecture Notes in Computer Science* **2162,** 349–363 (May 2001).
32. S.-B. Xu and L. Batina, "Efficient Implementation of Elliptic Curve Cryptosystems on an ARM7 with Hardware Accelerator," *Proceedings of the 4th International Information Security Conference (ISC'01)*; published in *Lecture Notes in Computer Science* **2200,** 266–279 (October 2001).
33. P. L. Montgomery, "Modular Multiplication Without Trial Division," *Math. Comp.* **44,** No. 170, 519–521 (1985).
34. Ç. K. Koç, T. Acar, and B. S. Kaliski, Jr., "Analyzing and Comparing Montgomery Multiplication Algorithms," *IEEE Micro* **16,** No. 3, 26–33 (June 1996).
35. Ç. K. Koç and T. Acar, "Montgomery Multiplication in GF(2^k)," *Designs, Codes, & Cryptog.* **14,** No. 1, 57–69 (April 1998).
36. A. Satoh and K. Takano, "A Scalable Dual-Field Elliptic Curve Cryptographic Processor," *IEEE Trans. Computers* **52,** No. 4, 449–460 (April 2003).
37. National Institute of Standards and Technology, "Digital Signature Standard (DSS)," FIPS PUB 186-2, January 2000; see *http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf*.
38. J. Postel, "Domain Name System Structure and Delegation," RFC 1591 (RFC1591), March 1994; see *www.faqs.org/rfcs/rfc1591.html*.

Received October 17, 2003; accepted for publication February 23, 2004; Internet publication September 17, 2004

Shigenori Shimizu *IBM Research Division, IBM Tokyo Research Laboratory, 1623-14 Shimo-tsuruma, Yamato-shi, Kanagawa-ken 242-8502, Japan (shimizus@jp.ibm.com).* Dr. Shimizu is a Senior Technical Staff Member working in computer architecture. He received B.E., M.S., and Ph.D. degrees, all in instrumentation engineering, from Keio University, Japan, in 1977, 1979, and 1983, respectively. He joined the research staff of the Japan Science Institute of IBM Japan (now the Tokyo Research Laboratory) in 1983. His areas of specialization have included VLSI architecture and design, multiprocessor architecture and design, and various embedded technologies. Dr. Shimizu also held several management positions, and currently leads the Systems and Technology Department in the Tokyo Research Laboratory.

Hiroshi Ishikawa *IBM Research Division, IBM Tokyo Research Laboratory, 1623-14 Shimo-tsuruma, Yamato-shi, Kanagawa-ken 242-8502, Japan (e04131@jp.ibm.com).* Mr. Ishikawa manages the Technology Services Group. He received B.S. and M.S. degrees in mechanical engineering from Waseda University in 1981 and 1983, respectively. He then joined the Tokyo Research Laboratory, where he has worked on electrical circuit and protocol design for various applications of robotics and wired and wireless communications.

Akashi Satoh *IBM Research Division, IBM Tokyo Research Laboratory, 1623-14 Shimo-tsuruma, Yamato-shi, Kanagawa-ken 242-8502, Japan (akashi@jp.ibm.com).* Dr. Satoh received B.S., M.S., and Ph.D. degrees in electrical engineering from Waseda University in 1987, 1989, and 1999, respectively. He joined the Tokyo Research Laboratory in 1989 and was involved in the research and development of digital and analog VLSI circuits. Dr. Satoh's current research interests include algorithms and architectures for data security and high-performance circuit implementations.

Toru Aihara *IBM Research Division, IBM Tokyo Research Laboratory, 1623-14 Shimo-tsuruma, Yamato-shi, Kanagawa-ken 242-8502, Japan (aihara@jp.ibm.com).* Mr. Aihara received a B.S. degree in electrical engineering in 1983 and an M.S. degree in electronic engineering in 1985, both from Tokyo University. He joined the Japan Science Institute of IBM Japan (now the Tokyo Research Laboratory) in 1985. Mr. Aihara was involved in research on low-power systems and wireless communication, and also in the standardization of the Bluetooth technology.