PAPER Special Section on Cryptography and Information Security

# Diversification of Processors Based on Redundancy in Instruction Set*

**Shuichi ICHIKAWA**[†,††a)], ***Member***, **Takashi SAWADA**[†], ***and*** **Hisashi HATA**[†], ***Student Members***

**SUMMARY** By diversifying processor architecture, computer software is expected to be more resistant to plagiarism, analysis, and attacks. This study presents a new method to diversify instruction set architecture (ISA) by utilizing the redundancy in the instruction set. Our method is particularly suited for embedded systems implemented with FPGA technology, and realizes a genuine instruction set randomization, which has not been provided by the preceding studies. The evaluation results on four typical ISAs indicate that our scheme can provide a far larger degree of freedom than the preceding studies. Diversified processors based on MIPS architecture were actually implemented and evaluated with Xilinx Spartan-3 FPGA. The increase of logic scale was modest: 5.1% in Specialized design and 3.6% in RAM-mapped design. The performance overhead was also modest: 3.4% in Specialized design and 11.6% in RAM-mapped design. From these results, our scheme is regarded as a practical and promising way to secure FPGA-based embedded systems.

***key words:*** *FPGA, embedded systems, instruction set, randomization, secure processor*

## 1. Introduction

Recently, software protection is becoming increasingly important. Infringement of copyright by software piracy, leakage of trade-secrets by reverse engineering, and attacks by malicious programs are all recognized as serious problems.

Forrest et al. [1] pointed out that the diversity of software could potentially increase the robustness of software systems. Although homogeneous systems have many advantages, such homogeneity makes malicious activities easier. For example, viruses often exploit the weakness of software to inject a binary code which alters the execution flow of a program (*injection attack*). Since an injection attack usually aims at a specific architecture, viruses easily spread among homogeneous systems.

Binary code injection would be more difficult, if each system had its own instruction set architecture. Such diversity would also obstruct piracy of software, because the software for a specific platform does not operate on other platforms. Reverse engineering of software would be also difficult without complete knowledge of the target architecture.

Although diversity has various advantages, processors of PCs and servers have been dominated by a small number of architectures. One reason is that users have profited more from monopoly than from diversity; another is that it is a logical consequence from the manufacturers' standpoint. Since recent software requires considerable processing power, competitive processors have to be manufactured with the latest process technology, which is very expensive. It is thus inevitable to mass-produce each processor, which naturally leads to oligopoly of processor architecture.

However, the situation is quite different for embedded systems, where requirements differ with their applications, and many systems are produced in relatively small quantities. Embedded systems do not necessarily require high-performance processors, but they often necessitate low-power and low-cost solutions. It is also very common to adopt SoC (System on Chip) technology, which integrates the processor and peripheral circuits into an LSI chip. As a result, various processors are used in embedded systems according to their purposes and applications.

Portability is not regarded as crucial in embedded software, because the latter is usually shipped as a part of the system. Rather, protection of software is a matter of critical importance. For example, copies might appear on the market if embedded software can be easily pirated. Valuable trade secrets might be unveiled and stolen, if reverse-engineering is easy. In manufacturing and transportation machinery, unauthorized modification of embedded software might result in serious accidents. Therefore, tamper resistance is very important and profitable in embedded systems.

Recent embedded systems often adopt FPGA (Field Programmable Gate Array) chips for implementation. FPGA is a reconfigurable logic LSI, which can maximally contain 100 million logic gates, 1 Mbyte of memory, and fast multipliers with the leading-edge process technology (90 nm process as of 2006). It is also popular to implement a microprocessor with the fabric of FPGA (*soft-core processor*). Since FPGA is flexible and suited to implement various systems on an item-by-item basis, it is well-suited to implement embedded systems, many of which are produced in small quantities. It is very easy to obtain diversity in embedded systems implemented with FPGA technology.

This study presents a method to diversify processors by utilizing the redundancy in instruction set architecture, which is particularly suited to embedded systems. Available

Manuscript received March 20, 2007.
Manuscript revised July 2, 2007.

[†]The authors are with the Department of Knowledge-based Information Engineering, Toyohashi University of Technology, Toyohashi-shi, 441-8580 Japan.
[††]The author is with the Intelligent Sensing System Research Center, Toyohashi University of Technology, Toyohashi-shi, 441-8580 Japan.

*This work was partially presented in SCIS2007 held in Nagasaki (January 2007).

a) E-mail: ichikawa@ieee.org

DOI: 10.1093/ietfec/e91–a.1.211

Copyright © 2008 The Institute of Electronics, Information and Communication Engineers

redundancies are estimated with some instruction sets, and the derived results are compared with the results of existing methods. The evaluation results of FPGA implementations are also shown for quantitative discussion of implementation cost and performance overheads.

## 2. Related Studies

There are many preceding studies on tamper-resistant hardware. Encryption of memory image is one of the typical approaches to avoid tapping and tampering. This approach was adopted by XOM (execute-only memory) [2], AEGIS [3], and SP-processor [4]. Another typical approach is to verify MAC (message authentication codes) of memory blocks to detect tampering. SPEF (secure program execution framework) [5] embeds encrypted, processor-specific constraints into each block of instructions at software installation time to verify their existence at run-time. BBST [6] maintains MACs of basic blocks for verification, while SAFE-OPS [7] integrates an FPGA-based secure hardware component for dynamic verification of MAC.

Dynamic memory encryption and MAC verification inevitably involve performance overhead caused by the increase of memory access latency. For XOM, one-time-pad encryption [8], [9] and memory predecryption [10] have been proposed to hide memory access latencies. To hide MAC verification latency for SPEF, Drinić and Kirovski [11] introduced overlapping of program execution and MAC verification, where instructions are speculatively executed and committed only after verifying the MAC value.

Forrest et al. [1] suggested increasing the robustness of computer systems by diversification and randomization. Shacham et al. [12] discussed the merits of address-space randomization, basically assuming software implementations. In connection with the diversification of instruction set architecture, instruction set randomization [13]–[15] has been studied. Though there are various levels and means to randomize an instruction set, their advantages are mostly common to this study: e.g., resistance to analysis, reverse engineering, and injection attacks. The following paragraphs provide a close look at instruction set randomization.

Barrantes et al. [13], [14] proposed to scramble instructions on memory by XORing instruction sequence with pseudo-random sequence. Scrambled instructions are unscrambled when they are fetched from external memory. Barrantes et al. named this scheme RISE (Randomized Instruction Set Emulation), though what is actually done here is *memory randomization* rather than *instruction set randomization*. Since they assumed the use of existing microprocessors, RISE adopts an emulator with binary-code translator for program execution, which involves significant performance overheads. This study, on the other hand, proposes a hardware-supported approach to instruction set diversification with modest overhead. Detailed evaluation results of our scheme will be presented in Sect. 6.

Kc et al. [15] proposed two methods for instruction set randomization. One is to scramble the instruction sequence by XORing a secret key, and the other is to randomly transpose all the bits within the instruction. The former method is similar to RISE, and is a memory scramble technique rather than instruction set randomization. The latter can be interpreted as instruction set randomization, if the length of instructions is fixed within the target architecture. The present study, however, presents a new method for instruction set randomization, which incurs less overhead than Kc's bit-transposition (as shown in Sect. 6). Our method can be applied to any architectures, while Kc's bit-transposition is not simply applicable to architectures of variable instruction lengths. Although our method and bit-transposition are both substitution ciphers, our approach is superior to Kc's methods because it has a larger degree of freedom. The quantitative evaluations of this aspect will be discussed in Sect. 4.

## 3. Processor Personality

Diversification of instruction set architecture may include any combinations of various diversifications; e.g., diversity of register sets, instruction formats, and data representations. However, all diversified processors have to guarantee a predefined specification (e.g., logic scale, operational frequency, power consumption, and reliability), which is quite difficult in general cases. Also, it practically means that we have to generate the custom software tools (e.g., OS kernel, assembler, linker, and compilers) automatically for each instruction set. Thus, such extreme approaches are practically impossible or too expensive.

In this study, as a more practical alternative, we propose to change the encoding of instructions without modifying the instruction formats. For an example, let us consider a processor $P_1$ that provides ADD instruction (opcode = 1) and SUB instruction (opcode = 2). Here, we can consider another processor $P_2$, which is a twin of processor $P_1$ except that the respective opcodes of ADD and SUB are 2 and 1 in $P_2$. Although the expressions of an instruction sequence are different for $P_1$ and $P_2$, they have exactly the same architecture including the length of instruction sequence, data representation, memory access, conditional branch, and exception handling. In the following discussion, the difference of $P_1$ and $P_2$ is referred to as *processor personality*. In other words, processor personality $P_2$ is a derivative of the original personality $P_1$.

The instruction sequence for a personality can be easily converted to the corresponding representation of another personality, and the derived software is as reliable as the original. Hence, we only have to prepare a simple binary-converter for each personality, and all other software tools can be shared via binary conversion. The logic scales and operational frequencies of $P_1$ and $P_2$ are expected to be comparable, because the architecture is common in both personalities. The evaluation results of this point are presented in Sect. 6.

From the hardware point of view, the logic circuit of instruction decoder is slightly different in $P_1$ and in $P_2$. By implementing the mapping of opcode with RAM/ROM, two

or more personalities can share the same hardware, which is preferable for manufacturing. It is also possible to implement a processor that accommodates multiple personalities by using large mapping RAM, where the processor personality can be switched dynamically. Although Barrantes [13], [14] and Kc [15] mentioned changing the way of randomization for each process, it is equally possible to provide similar functionality with mapping-RAM implementation of our scheme.

Another advantage of our approach is that an arbitrary subset of instructions may be selectively randomized. For example, it is possible to randomize privileged instructions, while leaving non-privileged instructions intact. In this case, the OS kernel becomes dependent on a specific personality, while any user programs stay portable among the personalities of the same architecture. It is also possible to randomize floating-point instructions and application-specific instructions (e.g., DSP instructions), while leaving integer instructions as they are. System vendors may provide high-performance applications that are dependent on a specific personality, while providing generic low-performance versions of the same functionality by using floating-point and application-specific operations emulated by integer operations. Such a framework might be beneficial to distribute commercial application programs. Needless to say, any programs become personality-dependent if all instructions are randomized.

Although our approach is best suited to soft processor cores on FPGA technology, it is also applicable to emulation technologies, such as Java VM [16] and Transmeta Code Morphing [17]. Our approach is completely orthogonal and cooperative with the existing randomization schemes of Barrantes et al. [13], [14] and Kc et al. [15]. Memory encryption and MAC verification may be simultaneously adopted with our scheme, making it both very practical and widely applicable.

## 4. Redundancy of Instruction Set

This section examines the number of personalities which are derivable from an instruction set architecture. If an instruction set architecture involves a large degree of freedom (or redundancy), it can yield a large number of personalities. Thus, the degree of freedom is regarded as a security measure of our scheme. The degree of freedom might also be interpreted as a factor in implementation cost, because it specifies the lower bound of information to designate a personality of an architecture. In the following evaluations, four instruction set architectures are examined: MIPS [18] (Release 1), an example of 32-bit architecture; Renesas SH-3 [19], a 16-bit architecture; Intel 8080 [20] and Java VM [16], 8-bit architectures.

Figure 1 illustrates the instruction formats of MIPS architecture [18]. The **opcode** field designates the instruction format and the operation of an instruction. In MIPS architecture, the length and the position of the opcode field are fixed. In R-type instructions, the **function** field is used with

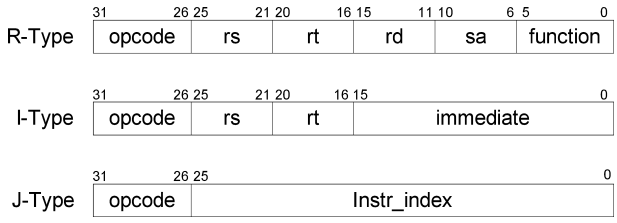


**Fig. 1** MIPS instruction format [18].

the opcode filed to designate a specific operation. The **rs**, **rt**, and **rd** fields designate operand registers, while the **immediate** field contains an immediate operand in I-type instructions.

Although the instruction format of MIPS is simple, it is not the case in general. In some architectures, the length of the opcode is variable, and part of opcode (or auxiliary field) appears at different positions in different formats. Thus, this study imposes the following restriction in the following evaluations; the opcode values are exchanged only among the same field of the same format. For example, the opcode values of J-type instructions (Fig. 1) might be interchangeable, while the opcode values of I-type in the original architecture are not considered for J-type instructions in a new personality. With I-type instructions, the values of function field are interchangeable only among the instructions of the same opcode value.

This restriction is not only natural from the hardware point of view, but rather required to evaluate various architectures in equal conditions. The results shown in this section should be hence considered moderate, and it might be possible to extract a larger degree of freedom from an instruction set by removing this restriction (i.e., by exchanging opcode values across instruction formats). Encoding and locations of operand descriptors might also be used for diversification, but they are not counted in this study.

The undefined opcodes in the instruction set specification are automatically excluded from the degree of freedom by the above restriction, because the instruction format is not defined for undefined instructions. Anyway, the undefined instructions do not appear in general programs, and do not contribute to increasing diversity.

Table 1 lists the evaluation results of the degree of freedom ($F$) of each instruction set. The corresponding information capacity $I$ is given by the equation $I = \lfloor \log_2 F \rfloor$, where $F$ is the degree of freedom. Table 2 summarizes the degrees of freedom for typical instruction classes in each architecture.

The degrees of freedom differ significantly among the four instruction sets, while the numbers of available instructions are comparable. A longer instruction format does not necessarily lead to a larger degree of freedom. Rather, in this study, larger diversity is obtained from an architecture with simple instruction formats, where a large number of instructions (opcodes) are defined for each format. For example, the opcodes of Java VM are 1-byte long for any kind of instructions, in which 201 instructions are defined. Con-

**Table 1** Redundancy of four instruction sets.

| | Number of instructions | Redundancy | Information [bit] |
|---|---|---|---|
| MIPS | 170 | 2.34e+166 | 553 |
| SH-3 | 188 | 1.63e+90 | 300 |
| 8080 | 111 | 2.34e+136 | 453 |
| Java VM | 201 | 1.59e+377 | 1253 |

**Table 2** Redundancy in various instruction classes.

| | Arithmetic | Data transfer | Branch | Control | Privileged |
|---|---|---|---|---|---|
| MIPS | 4.68e+21 | 1.29e+39 | 4.05e+18 | 2.00e+00 | 8.64e+03 |
| SH-3 | 6.53e+28 | 9.06e+25 | 1.15e+03 | 5.76e+03 | 2.40e+01 |
| 8080 | 2.30e+10 | 1.55e+25 | 3.05e+29 | 4.03e+04 | – |
| Java VM | 2.65e+32 | 8.50e+101 | 2.59e+22 | 1.00e+00 | – |

sequently, the degree of freedom becomes very large; i.e., $201! \approx 1.59 \times 10^{377}$. The great advantage of this scheme is that a simple architecture can yield a very large degree of freedom.

Kc et al. [15] proposed to obtain diversity by XORing a 32-bit secret key to instructions. Evidently, the degree of freedom of this method is limited to $2^{32} \approx 4.3 \times 10^9$. Kc et al. [15] also proposed to transpose all the bits randomly within a 32-bit instruction. The degree of freedom of this method is $32! \approx 2.63 \times 10^{35}$, and the corresponding information is approximately 118 bits. Although Kc's methods and ours are both substitution ciphers, our method can yield far larger diversity and is consequently more resistant to brute-force attacks than Kc's methods in any of the four architectures.

## 5. Design and Implementation

This section introduces sample designs of diversified processors of MIPS architecture. As a basis of the following discussion, a soft-core processor Plasma [21] was adopted. Plasma supports a subset of MIPS (release 1) instruction set and is publicly available as the VHDL source code, which is favorable for diversification. MIPS architecture is often adopted in actual embedded systems, and its simple instruction set is preferable for our experiments. The block diagram of Plasma is illustrated in Fig. 2.

The current version of Plasma defines 94 instructions, where 62 instructions are actually supported while 32 instructions are still unavailable. Since these are different from the undefined instructions mentioned in Sect. 4, all 94 instructions are used for randomization in the following experiments. The degree of freedom was thus calculated as $3.36 \times 10^{112}$ for Plasma, which corresponds to 373 bits of information.

The following five designs are examined in this study.

### 5.1 Original Design

The Original design is the original Plasma design, whose functional block diagram is illustrated in Fig. 2[†]. Plasma consists of six functional blocks: PC (program counter), IF (instruction fetch), ID (instruction decode), EX (execution), MA (memory access), and memory.

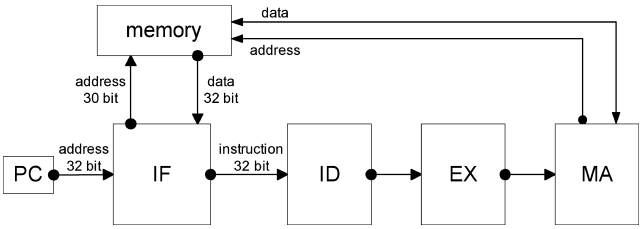


**Fig. 2** Block diagram of original design.

### 5.2 Specialized Design

The Specialized design designates the designs derived from the Original by randomly exchanging opcodes.

In the original Plasma source code, many literals are embedded (or *hard coded*) in VHDL source files. Since this is not favorable for our experiments, we modified the original VHDL files and added a new VHDL file that defines literals as constants. This modification enabled us to specialize the instruction set by preparing a new constant definition file. Each instance of Specialized design is thus generated through this file, which defines the literals randomly exchanged according to our rules.

### 5.3 RAM-Mapped Design

The RAM-mapped design implements the mapping of opcodes by RAM, consequently enabling changes of personality. The block diagram of the mapping part is illustrated in Fig. 3.

In RAM-mapped design, the opcode of a diversified processor is mapped to the original opcode by the Mapping block (Fig. 3), which is located in series between the IF and ID blocks. The opcode of a specialized instruction (op') is mapped to the original opcode (op) by RAM1, while RAM2 and RAM3 map the function field and the rt field, respectively[††]. The **repack** part does not include any logic function; it just reconstructs 32-bit instructions by concatenating bit fields. The repack block outputs the following three patterns of instructions.

**Case I:** The opcode field is mapped, while leaving other fields intact. Typically selected for J-type instructions.

**Case II:** The opcode and function fields are mapped, while leaving other fields intact. Typically selected for R-type instructions.

**Case III:** The opcode and rt fields are mapped, while leaving other fields intact. Typically selected for REGIMM instructions [18]. REGIMM instructions are a kind of I-type instruction, whose opcode value is 1. REGIMM instructions interpret the rt field as an auxiliary field.

[†]It should be noted that this diagram is different from the *physical* block diagram, which is found in Plasma document [21].

[††]These fields are illustrated in Fig. 1.

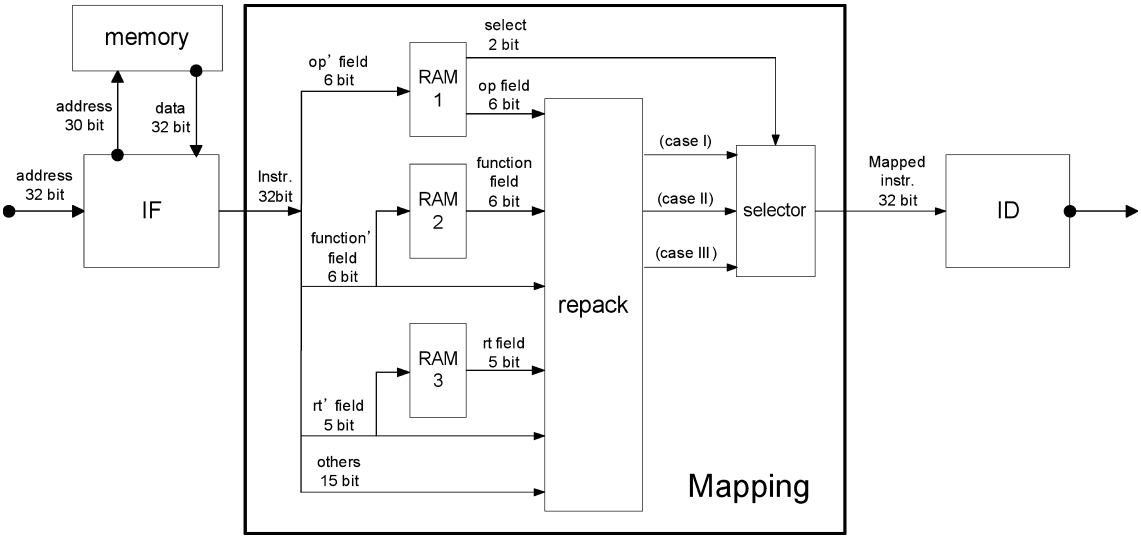


**Fig. 3** Partial block diagram of RAM-mapped design.

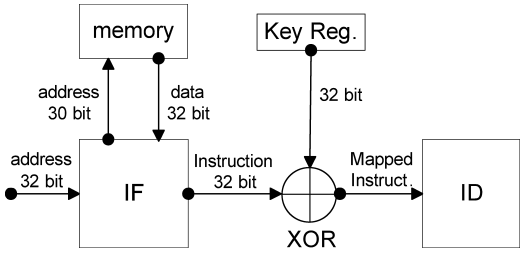


**Fig. 4** Partial block diagram of XOR design.

Typical examples of REGIMM instructions are BLTZ and BGEZ instructions.

The output selector in Fig. 3 selects the appropriate one from the above three cases, according to the opcode value. The selection is given by RAM1, since it is indexed by the opcode field.

As seen from the above descriptions, the Mapping block partially decodes instructions. Thus, the function of the Mapping block might be integrated in the ID block, because the interpretation of opcode is an intrinsic function of ID block. However, in the present study, we chose to make the Mapping block portable and independent from ID to keep the modification minimal. This decision was justified because our target architecture was simple. For more complicated architectures, the Mapping block should be integrated in the ID block to reduce the complexity and redundancy.

### 5.4 XOR Design

The XOR design includes a bitwise XOR function with a 32-bit key register between IF an ID blocks (Fig. 4), which literally implements the randomization process that was proposed by Kc et al. [15].

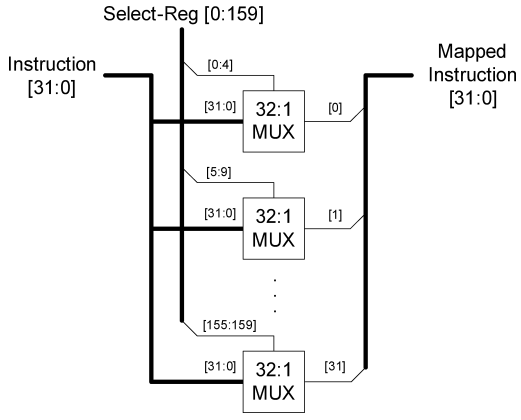


**Fig. 5** Design of bit-shuffling logic.

### 5.5 Bit-Shuffle Design

The Bit-shuffle design includes a bit-shuffling block, which is simply inserted between ID and IF blocks of the Original design. This bit-shuffling block was implemented with 32 multiplexers of 32 inputs and 160-bit register for 5-bit select inputs of 32 multiplexers, exactly as Kc et al. [15] proposed. The block diagram of the bit shuffling block is summarized in Fig. 5.

## 6. Evaluation

This section examines the implementation cost and performance overhead of diversified processors with FPGA technology. Xilinx Spartan3 FPGA [22] was adopted as the implementation platform in this section. The functions of diversified processors were actually verified on an evaluation board (Xilinx Spartan3 Starter Kit) with XC3S200 device.

Table 4 summarizes the design results of five designs, where the optimization option of logic synthesis was set

to "Speed" to maximize performance. The evaluation environment is summarized in Table 3. The unit of resources is "Slice" in Spartan3 FPGA, and Slices consist of logic resources (SliceL) and distributed memory resources (SliceM). Spartan3 FPGA also provides another kind of memory resource (BlockRAM), which is suited to data storage. For example, Mapping RAMs (RAM1, RAM2, and RAM3) in Fig. 3 were implemented with SliceM, while the main memory of Plasma was implemented with BlockRAM ("memory" in Fig. 2). Table 4 lists the number of Slice, SliceM, BlockRAM, and AT product of each design. AT product (Area-Time product) is defined by the product of logic scale (slices) and latency (cycle time), or the quotient of logic scale divided by operational frequency, which is often used as a measure of cost-effectiveness. Since the logic scale is a good measure of implementation cost, a smaller AT product naturally means lower cost per unit performance.

In Table 4, the result of Specialized design represents the average value of 100 randomly specialized processors. With regard to Slice, the average value was 2009 slices, where the minimum and maximum values were 1868 and 2117 slices, respectively. The operational frequencies were distributed between 31.1 an 37.9 MHz, which means the execution time might deviate maximally 10% from the average in a specialized design. The deviations of logic scale and execution time were modest, and seem acceptable for practical applications.

Though RAM-mapped is comparable to Specialized in logic scale, its operational frequency is 8.5% lower than Specialized. This is a logical consequence of our implementation, which has the Mapping block in series between IF and ID blocks (Fig. 3). The performance overhead is expected to be reduced, if the mapping logic is integrated in the ID block.

The usage of SLICEM is of interest. For example, Original and RAM-mapped designs use the same number of SLICEMs. RAM-mapped design has the Mapping block added to Original, and the Mapping block should include some SLICEM for mapping RAM. Therefore, it is natural for RAM-mapped design to use more SLICEM resources than Original, though the evaluation result is different. Another example is that a certain amount of SLICEM is included in Original, Specialized, XOR, and Bit-shuffle designs, though none of these designs uses SLICEM resources explicitly. The fact is that most SLICEM resources are automatically generated by logic synthesis software. All of the above phenomena are the consequence of the optimization process by logic synthesis, which utilizes memory resources to derive optimal results whenever possible[†].

The XOR design is expected to be larger than the Original design in logic scale, since the XOR design additionally includes a key register and a bitwise XOR circuit between IF and ID blocks of the Original design. Thus, it may seem strange that the logic scale of XOR is 1.25% smaller than that of Original in Table 4. However, it is only the consequence of the optimization option "Speed," which trades the logic scale for higher operational frequency. Table 5 summarizes the evaluation results with the optimization option "Area," which minimizes the usage of hardware resources. In Table 5, the XOR design shows a 3.0% increase of logic scale as we expected. Since the significant slowdowns were observed with the "Area" option, we have adopted the results with the "Speed" option (Table 4) in the above discussion.

The synthesis times of each design are almost comparable (Tables 4 and 5). Here, it should be noted that a Specialized design requires its synthesis time for each instance of diversified processors, while the other four designs require the synthesis time only once.

In conclusion, the overhead of diversification is insignificant. As shown in Table 4, Specialized design incurs

**Table 3** Evaluation environment.

| | |
|---|---|
| CPU | AMD Athlon XP 2500+ |
| Memory | 1 GB |
| OS | Windows XP SP2 |
| CAD software | Xilinx ISE 8.2i |
| Target device | XC3S2000 -4 FG456 |

[†]Any logic functions can be implemented with memory resources; this is the principle of contemporary FPGA devices.

**Table 4** Evaluation results of five designs (optimization option: Speed).

| | Slice (SliceM) | BlockRAM | Freq. [MHz] | AT prod. [slice·$\mu$s] | Synthesis [s] |
|---|---|---|---|---|---|
| Original | 1911 (128) | 4 | 35.3 | 54.1 | 437 |
| Specialized (avg.) | 2009 (128) | 4 | 34.1 | 58.9 | 479 |
| RAM-mapped | 1979 (128) | 4 | 31.2 | 63.4 | 461 |
| XOR | 1887 (128) | 4 | 34.2 | 55.2 | 494 |
| Bit-shuffle | 2139 (192) | 4 | 33.2 | 64.4 | 522 |

**Table 5** Evaluation results of five designs (optimization option: Area).

| | Slice (SliceM) | BlockRAM | Freq. [MHz] | AT prod. [slice·$\mu$s] | Synthesis [s] |
|---|---|---|---|---|---|
| Original | 1481 (128) | 4 | 19.3 | 76.6 | 302 |
| Specialized (avg.) | 1522 (128) | 4 | 17.9 | 85.0 | 336 |
| RAM-mapped | 1526 (128) | 4 | 17.4 | 87.7 | 380 |
| XOR | 1513 (128) | 4 | 18.1 | 83.6 | 369 |
| Bit-shuffle | 1675 (192) | 4 | 18.1 | 92.5 | 380 |

a 5.1% increase of logic scale with a 3.4% loss of operational frequency, while RAM-mapped design incurs a 3.6% increase of logic scale with a 11.6% loss of operational frequency. On the other hand, Kc's XOR design incurs a 3.1% loss of operational frequency with almost the same amount of logic circuit as Original. Kc's Bit-shuffle requires more logic than Specialized and RAM-mapped designs (11.9% increase over the Original) with a modest loss of operational frequency (5.9% loss over the Original). Considering the large difference in the degree of freedom shown in Sect. 4, Specialized and RAM-mapped designs are regarded as superior to XOR and Bit-shuffle designs.

## 7. Discussion

### 7.1 Conditions and Limitations

In many embedded systems, cost is a matter of highest priority. The logic scale and memory capacity of the system are thus tightly restricted in the name of price competitiveness. This often results in modest performance of processor with low clock frequency, small (or sometimes no) cache memory, and low-bandwidth of main memory. Such savings are accumulated cent by cent over every component of the system. In such systems, any possible overhead must be avoided in both clock cycles and memory usage in order to satisfy the system requirements.

Memory encryption (e.g., XOM [2]) is a reliable approach to secure systems, which is based on cryptography theory. A serious problem of this approach is that cryptographic circuit requires a substantial amount of resources. For example, Rouvroy et al. [23] presented a very compact AES circuit, which requires 163 slices on a XC3S50 device for 208 Mbps throughput. Good and Benaissa [24] presented a high-performance AES circuit, which requires 17425 slices on XC3S2000 for 25 Gbps throughput. Although it is highly dependent on implementation details, the resource requirement of a practical AES circuit is far from negligible in general cases[†]. Another problem is that memory encryption inevitably incurs increased memory access latency, which leads to substantial degradation of performance. This problem might be avoided by implementing a large cache memory, which would be too expensive for cost-sensitive embedded applications. Thus, the memory encryption approach would be better suited for security-oriented applications, while our approach is more suited to cost-sensitive embedded systems.

In case of RISE [13], [14], a binary code is XORed with the pseudorandom sequence that is as long as the whole program text. RISE decodes the whole encrypted binary code at loading time, and stores the plain binary code in the internal memory for execution. Though this mechanism could be implemented by hardware to avoid loss in performance, it requires a large internal memory that could contain the whole program image. This will raise the system cost substantially. On the other hand, our approach is free from such problems.

### 7.2 Possible Attacks

Both our scheme and Kc's XOR scheme are a kind of simple substitution cipher, which preserve the statistical properties of the original instruction sequence. Such properties could be exploited by frequency analysis in a ciphertext-only attack.

Possible attacks on the Kc's XOR scheme might be conducted as follows. The opcode of MIPS instruction is 6-bit long, which resides at the same position in all instructions (Fig. 1). This practically means that the effective key length is only six bits for opcode, and the corresponding part of the key can be analyzed independently by analyzing the frequencies of opcode values. Once the opcode (or the corresponding part of the key) is revealed, the instruction format is determined. R-type instructions are particularly preferable for frequency analysis, since it consists of short fields of 5- or 6-bits long, each of which can be analyzed independently and easily. The burden of analysis will be much alleviated by heuristics, as shown in the following examples [18].

- Specific fields are set to be constant in some instructions. For example, the **sa** field is always set to zero in ADD (add word) and SUB (subtract word) instructions.
- In most cases, the **rs** field of JR (jump register) instruction is 31 in MIPS binary code, because JAL (jump and link) instruction automatically saves the return address into a specific register (R31).

Although the XOR design might look good in both logic scale and operational frequency in Tables 4 and 5, it is vulnerable and inadequate for practical applications.

Though our scheme also suffers from frequency analysis, it is not as simple as in the XOR scheme. In the XOR scheme, any instruction can be decoded, once the key is identified by frequency analysis. On the other hand, in our scheme, each opcode value has to be identified independently to a specific instruction. Thus, even if one instruction is identified, other instructions remain unknown. This difference is a logical consequence of the larger degree of freedom of our scheme.

Since a simple substitution cipher is not reliable enough, transposition or permutation is often adopted together with substitution. To complement our scheme, it is worth considering transposing instruction bits randomly for each instance of diversified processors to obfuscate the position of the opcode field. This will raise the degree of freedom by 118 bits, as described in Sect. 4. Bit transposition requires no additional cost, because it is simply implemented by wiring without logic gates. Though the Bit-shuffle design in Sect. 6 incurs much overhead, it was caused by the multiplexers needed to change bit positions arbitrarily.

It is also worth considering XORing instruction word

[†]More information on cryptographic circuits on FPGA can be found in a comprehensive survey by Wollinger et al. [25]

with a constant, randomly generated for each instance of diversified processors. This will adjust the distribution of ones and zeros in instruction bits, consequently making it more difficult to analyze a personality. This will raise the degree of freedom by 32 bits with a minimal increase of logic scale (avg. 16 inverters).

Another concern is that an attacker may infer a personality from another personality. If an attacker managed to acquire two corresponding object images of the same instruction sequence for two different personalities, it is possible for the attacker to make the list of the corresponding opcodes of two personalities by examining the differences of two object images. If the common object code was long enough to include all (or most) instructions of target architecture, the derived list will enable the attacker to convert arbitrary object code from one personality to the other personality. Moreover, if one personality is cracked by the attacker, the other personality might be inferred with this list.

There are two ways of avoiding this problem. One is to diversify the instruction sequence for each system, and the other is to disturb the object code comparison. Various techniques for software diversification have been already discussed by Forrest et al. [1]; e.g., permuting the order of basic blocks yields a large degree of freedom [26], [27]. Processor diversification does not conflict with software diversification techniques; it is rather recommended to adopt these two together, whenever necessary. On the other hand, the encryption of memory image (e.g., XOM [2] and RISE [14]) will prevent attackers from comparing binary codes of two personalities. All these techniques might be adopted together, carefully considering the trade-offs between security and cost on a case-by-case basis. It should be noted that our scheme alone cannot provide excellent security.

Although processor diversification cannot resolve this problem individually, it never means that processor diversification is useless. Software diversification itself does not prevent the attacker from software piracy, but it is processor diversification that disturbs software piracy. It is also processor diversification that protects the system from hijacking, because the attacker has to analyze the personality before writing new malicious software for the target system. Meanwhile, software diversification serves to protect diversified processors from the analyses by malicious attackers.

### 7.3 Securing Configuration

In case of a diversified processor, it is difficult to analyze or plagiarize the software, even if memory images are stolen. However, if the processor design itself is stolen and duplicated, the software may be utilized as a black box. Though it is impossible to write new software without knowing the instruction set, it might be analyzed from the stolen processor design. This poses a very serious problem for FPGA implementation, because it is very easy to duplicate a processor by tapping or stealing the configuration data. Since it is particularly easy if the configuration data are stored in external memory devices, securing configuration data is essential for our objective.

Recent FPGA devices often support design security features. For example, LatticeXP FPGA [28] provides an on-chip non-volatile memory for configuration data, which secures the design against tapping. Altera Stratix II FPGA [29] supports AES encryption of configuration data, where the 128-bit key is kept secure in an on-chip non-volatile memory. The encrypted configuration data cannot be analyzed nor plagiarized without the secret key, even if the configuration data are stored in external memory. Actel ProASIC3 FPGA [30] provides both an on-chip non-volatile memory and 128-bit AES encryption for configuration data. Our scheme is best suited to such devices that provide design security features.

In XOM [2] and RISE [13], [14], the system is protected by a key. If the key is stolen by an attacker, the software may be copied, analyzed, and modified freely by the attacker. The attacker might take control of the whole system through the key. In our scheme, the configuration data of FPGA must be protected against attackers. However, even if the configuration data are stolen by any means, they are not as portable as software. Since the configuration data are device-dependent, they only work on the original target device. The configuration data are also dependent on the design of the circuit board, since they include pin assignment information to interface with various external circuits. Thus, the configuration data are not as convenient as cryptographic keys for attackers.

Though it is theoretically possible to analyze the instruction set architecture by reverse-engineering the configuration data, it would take a substantial amount of work. Moreover, such effort would have to be repeated for each personality. Considering all these things, our scheme does not seem an easy target for attackers in practical situations. RAM-mapped design might seem relatively easy to analyze, but it would be more resistant to reverse-engineering if the Mapping block is integrated with the ID block.

## 8. Conclusion

This study presented a new method to diversify processor architecture by changing the encoding of opcode, while preserving the function of the original instruction set architecture. Our method provides genuine *instruction set randomization*, which has not been realized to date in preceding studies. Our method also provides a larger degree of freedom, and thus it is more secure than the preceding studies.

The evaluation results on FPGA indicated that the overhead of our scheme is modest; with the RAM-mapped design, the overhead was 3.6% in logic scale and 11.6% in performance over the original Plasma. We thus conclude that processor diversification based on instruction set redundancy is a promising scheme for software protection in embedded systems, particularly when implemented with FPGA technology.

It is possible and desirable to adopt our method in combination with the preceding methods for more security, since

our method is independent of, yet compatible with, them. Although this study presented a new method of diversifying instruction set architecture, it has specific preconditions and limitations, like any other scheme. Designers must adopt a number of schemes together to secure a system in practical situations. This must be done very carefully and on a case-by-case basis.

## Acknowledgments

The authors are grateful to the editor and anonymous reviewers for their valuable comments and suggestions. The authors are also grateful to Prof. Hiroaki Takada for his valuable comments at the beginning of this work.

This work was partially supported by a Grant-in-Aid for Scientific Research from the Japan Society for the Promotion of Science (JSPS). Support for this work was also provided by the 21st Century COE Program "Intelligent Human Sensing" from the Ministry of Education, Culture, Sports, Science and Technology of Japan.

### References

[1] S. Forrest, A. Somayaji, and D. Ackley, "Building diverse computer systems," HotOS'97: Proc. 6th Workshop on Hot Topics in Operating Systems (HotOS-VI), p.67, IEEE Computer Society, Washington, DC, USA, 1997.

[2] D.L.C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell, and M. Horowitz, "Architectural support for copy and tamper resistant software," ASPLOS-IX: Proc. Ninth International Conference on Architectural Support for Programming Languages and Operating Systems, pp.168–177, ACM Press, New York, NY, USA, 2000.

[3] G.E. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas, "AEGIS: Architecture for tamper-evident and tamper-resistant processing," ICS'03: Proc. 17th Annual International Conference on Supercomputing, pp.160–171, ACM Press, New York, NY, USA, 2003.

[4] R.B. Lee, P.C.S. Kwan, J.P. McGregor, J. Dwoskin, and Z. Wang, "Architecture for protecting critical secrets in microprocessors," ISCA'05: Proc. 32nd Annual International Symposium on Computer Architecture, pp.2–13, IEEE Computer Society, Washington, DC, USA, 2005.

[5] D. Kirovski, M. Drinić, and M. Potkonjak, "Enabling trusted software integrity," ASPLOS-X: Proc. 10th International Conference on Architectural Support for Programming Languages and Operating Systems, pp.108–120, ACM Press, New York, NY, USA, 2002.

[6] M. Milenković, A. Milenković, and E. Jovanov, "A framework for trusted instruction execution via basic block signature verification," ACM-SE 42: Proc. 42nd Annual Southeast Regional Conference, pp.191–196, ACM Press, New York, NY, USA, 2004.

[7] J. Zambreno, A. Choudhary, R. Simha, B. Narahari, and N. Memon, "SAFE-OPS: An approach to embedded software security," ACM Trans. Embedded Computing Sys., vol.4, no.1, pp.189–210, 2005.

[8] G.E. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas, "Efficient memory integrity verification and encryption for secure processors," MICRO 36: Proc. 36th Annual IEEE/ACM International Symposium on Microarchitecture, p.339, IEEE Computer Society, Washington, DC, USA, 2003.

[9] J. Yang, Y. Zhang, and L. Gao, "Fast secure processor for inhibiting software piracy and tampering," MICRO 36: Proc. 36th Annual IEEE/ACM International Symposium on Microarchitecture, p.351, IEEE Computer Society, Washington, DC, USA, 2003.

[10] B. Rogers, Y. Solihin, and M. Prvulovic, "Memory predecryption: Hiding the latency overhead of memory encryption," ACM/SIGARCH Comput. Archit. News, vol.33, no.1, pp.27–33, 2005.

[11] M. Drinić and D. Kirovski, "A hardware-software platform for intrusion prevention," MICRO 37: Proc. 37th Annual IEEE/ACM International Symposium on Microarchitecture, pp.233–242, IEEE Computer Society, Washington, DC, USA, 2004.

[12] H. Shacham, M. Page, B. Pfaff, E.J. Goh, N. Modadugu, and D. Boneh, "On the effectiveness of address-space randomization," CCS'04: Proc. 11th ACM Conference on Computer and Communications Security, pp.298–307, ACM Press, New York, NY, USA, 2004.

[13] E.G. Barrantes, D.H. Ackley, T.S. Palmer, D. Stefanovic, and D.D. Zovi, "Randomized instruction set emulation to disrupt binary code injection attacks," CCS'03: Proc. 10th ACM Conference on Computer and Communications Security, pp.281–289, ACM Press, New York, NY, USA, 2003.

[14] E.G. Barrantes, D.H. Ackley, S. Forrest, and D. Stefanović, "Randomized instruction set emulation," ACM Trans. Inf. Syst. Secur., vol.8, no.1, pp.3–40, 2005.

[15] G.S. Kc, A.D. Keromytis, and V. Prevelakis, "Countering code-injection attacks with instruction-set randomization," CCS'03: Proc. 10th ACM Conference on Computer and Communications Security, pp.272–280, ACM Press, New York, NY, USA, 2003.

[16] T. Lindholm and F. Yellin, Java Virtual Machine Specification, Addison-Wesley Longman Publishing Co., Boston, MA, USA, 1999.

[17] J.C. Dehnert, B.K. Grant, J.P. Banning, R. Johnson, T. Kistler, A. Klaiber, and J. Mattson, "The Transmeta Code Morphing™ Software: Using speculation, recovery, and adaptive retranslation to address real-life challenges," CGO'03: Proc. International Symposium on Code Generation and Optimization, pp.15–24, IEEE Computer Society, Washington, DC, USA, 2003.

[18] MIPS Technologies, Inc., MIPS32™ Architecture for Programmers (Volume I), June 2003. Rev.2.00.

[19] Renesas Technology, SH-3/SH-3E/SH3-DSP Software Manual, May 15 2006. REJ09B0317-0400.

[20] Intel Corp., Intel 8080A/8080A-1/8080A-2 8-bit N-channel Microprocessor, Nov. 1986.

[21] S. Rhoads, "Plasma - most MIPS I (TM) opcodes: Overview," Nov. 2006. http://www.opencores.org/projects.cgi/web/mips/

[22] Xilinx, Inc., Spartan-3 FPGA Family: Complete Data Sheet, 26 April 2006. DS099 (v2.1).

[23] G. Rouvroy, F.X. Standaert, J.J. Quisquater, and J.D. Legat, "Compact and efficient encryption/decryption module for FPGA implementation of the AES Rijndael very well suited for small embedded applications," Proc. ITCC 2004, vol.2, pp.583–587, IEEE Computer Society, 2004.

[24] T. Good and M. Benaissa, "AES on FPGA from the fastest to the smallest," Proc. CHES 2005, pp.427–440, LNCS 3659, Springer, 2005.

[25] T. Wollinger, J. Guajardo, and C. Paar, "Security on FPGAs: State-of-the-art implementations and attacks," ACM Trans. Embedded Computing Sys., vol.3, no.3, pp.534–574, 2004.

[26] K. Hattanda and S. Ichikawa, "The evaluation of Davidson's digital signature scheme," IEICE Trans. Fundamentals, vol.E87-A, no.1, pp.224–225, Jan. 2004.

[27] K. Hattanda and S. Ichikawa, "Redundancy in instruction sequences of computer programs," IEICE Trans. Fundamentals, vol.E89-A, no.1, pp.219–221, Jan. 2006.

[28] Lattice Semiconductor, LatticeXP Family Handbook, Feb. 2007.

[29] Altera Corp., Stratix II Device Handbook (Volume 1), 2006.

[30] Actel Corp., ProASIC3E Flash Family FPGAs, April 2007.

**Shuichi Ichikawa** received his D.S. degree in Information Science from the University of Tokyo in 1991. He has been affiliated with Mitsubishi Electric Corporation (1991–1994), Nagoya University (1994–1996), and Toyohashi University of Technology (since 1997). Currently, he is an associate professor of the Department of Knowledge-based Information Engineering of Toyohashi University of Technology. His research interests include parallel and distributed processing, high-performance computing, and custom computing systems. He is a member of ACM, IEEE, and IPSJ.

**Takashi Sawada** received his B.E. degree in 2006 from the Department of Knowledge-based Information Engineering of Toyohashi University of Technology. Presently, he is studying for his master's degree at that institution.

**Hisashi Hata** received his B.E. degree in 2007 from the Department of Knowledge-based Information Engineering of Toyohashi University of Technology. Presently, he is studying for his master's degree at that institution.