Chinese Journal of Electronics
Vol.25, No.5, Sept. 2016

The Design and Implementation of Embedded Security CPU Based on Multi-strategy*

LI Dongfang[1,2], ZHAN Xin[1,3], TONG Qiaoling[1], ZOU Xuecheng[1] and LIU Zhenglin[1]

(1. *School of Optical and Electronic Information, Huazhong University of Science and Technology, Wuhan 430074, China*)
(2. *Beijing Institute of Computer Technology and Application, Beijing 100854, China*)
(3. *Department of Electrical and Computer Engineering, Texas A&M University, Texas 77843, USA*)

Abstract — Control flow monitoring, information flow tracking and memory monitoring are the three main solutions to enhance the security of embedded system at the hardware architecture level. However, most of the current studies about the security of embedded system consider the above solutions in separate dimensions rather than a combined effort. We start from the operation model at the instruction level, and propose a security multi-strategy which combines information flow tracking and memory monitoring by studying the security operating mechanism of embedded system. As a hardware approach this strategy extends the embedded processor architecture with additional security defense control. The experimental results show this multi-strategy is more effective and can detect more malicious attacks than a single solution. The effectiveness of our proposed security multi-strategy has been verified in a Field programmable gate array (FPGA) prototype platform based on a customized Leon3 microprocessor.

Key words — Information flow security, Taint tracking, Memory monitor module, Embedded processor architecture.

I. Introduction

With the widely use of embedded systems, the security issues of embedded processors attract the increasing attentions. So far the research on embedded processor security mainly focuses on control flow monitoring, information flow tracking and memory monitoring[1]. However, most of the existing work considers them as separate dimensions, and very little work has proposed integrated approach and addressed multiple dimensions at the same time, which motivates our work.

Information flow tracking[2−7], also known as information flow control or taint tracking, is an important security policy. Dynamic information flow tracking (DIFT)[8] tags the untrusted information as tainted one and tracks its propagation in a security system. The DIFT appends every word in the system memory with a label, and tags new information coming from the untrusted one as tainted information. The security system will generate a security exception, in the case of tainted information which is used in a possible insecure way, such as running a tainted Structured query language (SQL) instruction or releasing a tainted pointer. Actually, many researches on information flow tracking have been done.

Information flow monitoring focused on tracking the flow of the external data into the processor (*e.g.* data from the General purpose input output (GPIO), serial ports, and networks), which can help prevent the illegal operations caused by these external data or program, such as stealing users' private information stored in the system. However, the information flow monitoring mechanism doesn't make a detailed analysis of the security of external data or programs. It only decides which data taint needs to propagate, and decides which data needs to be checked when checking the taint. Although the mechanism of information flow monitoring can detect some common attacks, it may result in the high false positive rate for other safety program in the system. In addition, in order to detect a certain type of attacks, it needs to configure a Taint propagation register (TPR) and a Taint detection register (TDR), if the type of attack changes, both of them need to be changed accordingly, which undoubtedly limits its flexibility.

Memory monitoring mechanism[9−13] achieves the purpose of detecting malicious attacks by protecting data space when the program runs and preventing malicious code from unauthorized modification of data space of a program. The program's data space includes stack sec-

*Manuscript Received Jan. 10, 2014; Accepted Feb. 1, 2016. This work is supported by the National Natural Science Foundation of China (No.61176026, No.61376026).

DOI:10.1049/cje.2016.08.040

tion, heap section, global data section and text section. Implementing memory monitoring strategy in the embedded processor can prevent many common buffer overflow attacks, such as stack overflow attacks and heap overflow attacks. There are several hardware-based methods of memory monitoring.

Memory monitoring needs to make a detailed analysis about the security of a program itself, including the type of instructions executed and the boundary information of the program's data space, which determine whether the instructions executed have threats on the program's data space. However, the process of compiling source code written by advanced programming language into machine instructions has a great relationship with the type of compiler. For a given passage of source code written by advanced programming language, the machine instruction compiled by different compiler may be different. Then the result analyzed by the memory monitoring module may also be different, which may lead to a high false positive rate and a high false negative rate. The above analysis shows that, in embedded systems, a single memory monitoring strategies is not enough to prevent all the malicious attacks.

In summary, the above three methods can improve the security of embedded processors with their own characteristics and advantages/disadvantages. Based on the above analysis, we give full consideration to their own strengths of information flow tracking and memory monitoring and combine them together. We design the information flow monitoring by modifying the Register transfer level (RTL) code of the kernel Integer unit (IU) and adding the TCR at the kernel IU. The information flow monitoring provides the functionality of classifying all kinds of attacks, capability of flexibly programming security policies, and capability of simultaneous multi-attack defending at very low cost. We implement the memory monitoring by adding a hardware module which runs in parallel with the embedded processor and it can effectively detect common buffer overflow attacks. Finally, we mapped our design to an FPGA development board and developed a prototype system. In order to make better use of these two methods, we adjust the security level of information flow tracking. Experimental results show that compared with a single strategy of information flow tracking and single strategy of memory monitoring, our multi-strategy can effectively detect more kinds of attacks at run time, which takes advantages of both the information flow tracking and memory monitoring, then enhances the overall security of embedded systems.

II. Architecture

As our design is a combination of information flow tracking and memory monitoring, the architecture of our design is divided into two parts: architecture of information flow tracking and architecture of memory monitoring.

1. Architecture of information flow tracking

By following the idea of DIFT, the embedded processor kernel is extended with taint spreading tags in order to provide taint tracking in a propagation environment. Every word is appended with 4 tag bits as specified in the Table 1. Because of the added tag bits, several modifications are applied in the hardware architecture, such as 4 bits extension in all registers, caches, memories and data bus for the taint propagation. When the Central processing unit (CPU) is running the instruction pipeline, all the data is attached with a 4 bits tag for tracking the propagation of data source. The tag bits extended in bus and memory are added on the Most significant bit (MSB) of data.

Table 1. Research subjects

Tag bits	Definition
Tag [0]	Taint mark bit: '1': The data is untrusted; '0': The data is trusted.
Tag [2:1]	Threat level classification bits: '00': The data has no threat; '01': The data is in a low threat level; '10': The data is in a middle threat level; '11': The data is in a high threat level.
Tag [3]	Sensitive information mark bit: '1': The data is sensitive; '0': The data is insensitive.

The process of taint tracking contains three tasks, namely the taint marking, taint propagation and taint checking. We propose to use threat level classification depending on the processor's actions to the external data. We consider that accuracy of the taint tracking can be enhanced when the system is operating and all abnormal behaviors can be internally monitored. We set the propagation logic as all-propagation, all taints will pass the all-propagation logic.

2. Architecture of memory monitoring

The amount of buffer information obtained through dynamic monitor decides the defending policy of the hardware security module. The buffers defined in the program are in the data space of the program.

A typical program data space as shown in Fig.1 includes stack section, heap section, global data section and text section. The local pointers, variables and local arrays defined in the program are stored in the stack section. A buffer defined by a memory allocated function such as malloc belongs to heap section. The global pointers, variables and global arrays defined in the program are saved in the global data section. Buffer overflow attack usually happens in these three sections, and the attack to stack section is the most frequent one. The program's binary executable code is stored in text section. The text section is read-only, so it is hard to be rewritten and it has a good resistance to tampering attacks.

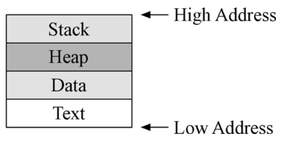

Fig. 1. Diagram of a typical program data space

1) Stack segment protection

As to stack segment protection, we reduce the range protected and only protect return address and stack pointers which have relatively fixed position in stack and are frequently attacked in stack.

2) Heap segment protection

As the accessibility of bound information in the stack section is limited, which makes a complete data protection of the stack data space impossible, the situation in heap section is totally different. The heap memory is allocated by a special function such as malloc dynamically. From the view of hardware, we can recognize the function malloc from its characteristic instructions and then obtain the buffer's start address and length information from the function's parameters, which can be used to completely prevent buffer overflow in the heap segment.

3) Data segment protection

The global variable, global array, static variable and static array are all stored in the global data space. The data that decides the executing direction of the program is called control data, such as the return address and function pointer, *etc*. If an attacker wants to get control of the program, he must modify the control data first. The control data is the address of certain instruction in the program, so it is stored in a word memory unit. The following operation to the memory unit is supposed to be word operation accordingly. The operation to a memory unit related to a character or a string array is byte operation. Usually, an attacker overflows a string array by changing the content of adjacent memory units. If the memory units happen to store a function pointer and the operation on it is a byte operation which is inconsistent with the previous operation mode to the memory unit, then an attack can be detected. We will construct our hardware protection module based on the above security policy.

III. Hardware Implementation

In order to validate the effectiveness and measure the performance of the combination of information flow tracking and memory monitoring, we use the LEON3 processor[14] as our prototyping platform, which is a 32 bits processor with the SPARC-V8 architecture.

We implement information flow monitoring and memory monitoring respectively by modifying the RTL code of the kernel IU, adding the TCR at the kernel IU and adding a hardware module which runs in parallel with the embedded processor. Fig.2 shows a simplified diagram of the hardware of our design.

To integrate the information flow monitoring function, we modified the open-source RTL codes of the LEON3 processor to add hardware support for appending 4 bits tags to the registers/caches, taint check registers and taint propagation logic. Besides, we extended the Advanced microcontroller bus architecture (AMBA) bus to be compatible with our tagged storage units, and provided the taint check process with a new exception generation mechanism. The tag marking, propagation and checking functions are added to the AMBA bus, Arithmetic-logic unit (ALU) and the pipeline's exception stage, respectively. When the external data transferred on the AMBA bus, it is labeled by the tag, and tag [0] is set to 1.

Fig.2 also shows the hardware architecture of memory monitor. The monitor module intercepts instructions on the data path between cache and IU unit and obtains security information needed for the monitoring from the pipeline. The memory monitor module includes three parts: stack protection module, heap protection module and global data protection module. The address decoder transmits target addresses of memory operation instructions to the corresponding hardware protection module according to the data space the address belongs to. We can see that the whole memory monitor module can operate in parallel with the processor completely, which makes the performance penalty to the overall system very small.

IV. Security Evaluation

To verify the effectiveness and quantify the performance overhead of our proposal defending against malicious software attacks, we have used two types of malicious attacks respectively in the verification process, which are stack overflow attack and heap overflow attack.

To make the two methods compatible with each other and achieve complementary advantages of both of these two methods, we have reduced the security level of information flow tracking. Some missed attacks can pass to memory monitoring module to detect, which not only reduces the false positive rate of information flow tracking, but also does not increase the false negative rate of the whole system. Verification results show that the combination of information flow monitoring and memory monitoring can effectively defend against a variety of attacks at run time.

1. Stack overflow attack

The buffer overflow attack model shown in Fig.3 has been considered in one of our experimental setup.

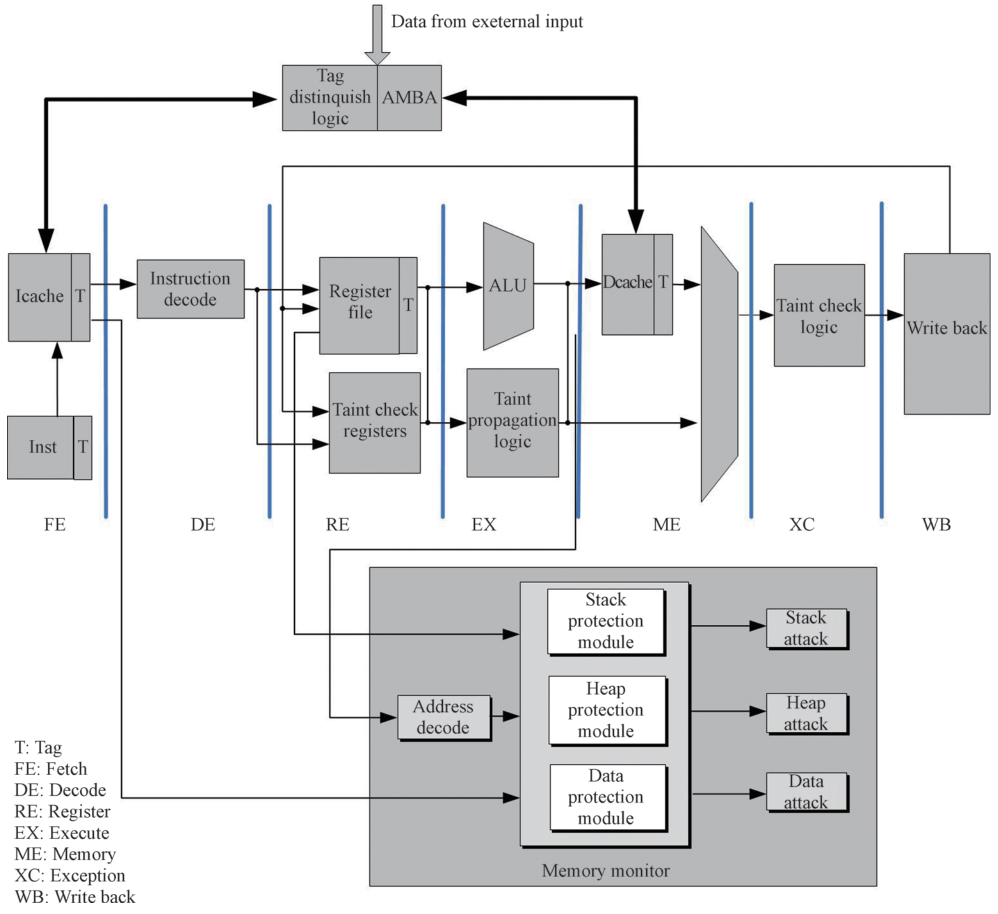

Fig. 2. Diagram of the hardware of our design

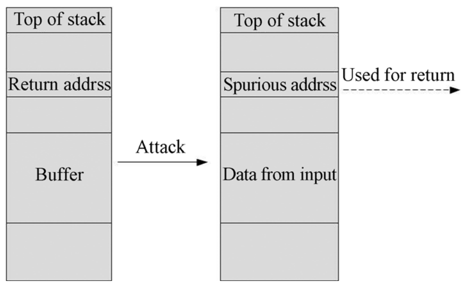

Fig. 3. Diagram of buffer overflow attack model

External data can be transmitted into the system through various channels, such as UART, GPIO, keyboard and ethernet. For our experimental setup, the taint tracking strategy mainly considers external data from the UART. The main segment of the attack code is shown in Fig.4.

Now we give an analysis of how information flow tracking and memory monitoring detect this attack. For information flow tracking, this attack modifies the return address through the buffer overflow. The taint check logic monitors the data flows when the processor is operating. After the return addresses are covered by the tainted data during the attacks, an exception is generated.

Memory monitoring can also easily detect above stack overflow attacks. There is a virtual instruction execution unit in the memory monitoring module. By inspecting the bound of stack space of a program, memory monitoring module can easily detect that the return address of function() has been covered illegally, and then detect the stack overflow attack.

```
int sum_2 (int a, int b)
{
    char buffer [15];
    char shell [80];
    int i;
    …
    for(i=0; i<80; i++)
      buffer [i] = shell [i];
    return c;
}
void spurious_attack()
{
    while(1)
    {
      console_print_string ("AttackSuccessful!!!");
    }
}
```

Fig. 4. The main segment of stack overflow attack code

2. Heap overflow attack

Heap overflow attack is one of the common attacks, but also belongs to buffer overflow attacks. In our experiments, we also used a heap overflow attack code to carry out the attack test on the experimental platform we built. The major difference is that this malicious code is not input from the external interface, but has been written in the Synchronous dynamic random access memory (SDRAM) of the system before the system is running.

The code of heap overflow attack shown in Fig.5 below has been simplified for better presentation. We retained the key code of this type of attack as follows. The *main*() function dynamically allocate two heap blocks, the starting address are buf1 and buf2. The distance between $buf1$ and $buf2$ is *BUFFER_DISTANCE*. First, it initializes $buf2$ by calling the *memset* function, and assigning "A" to $buf2$. Second, initializes $buf1$ by calling the *memset* function, but the length of assignment is *BUFFER_DISTANCE+OVERSIZE* which is longer than that of $buf1$. Because memset function does not check the boundary of $buf1$ and $buf2$, it caused a heap overflow and covered the data of $buf2$.

```
# include<studio.h>
#define BUFSIZE 16
#define OVERSIZE 8
main()
{
   unsigned long BUFFER_DISTANCE;
   char *buf1 = (char*)malloc(BUFSIZE);
   char *buf2 = (char*)malloc(BUFSIZE);
   BUFFER_DISTANCE = (unsigned long) buf2 - (unsigned long buf1;
   printf("buf1=%p, buf2=%p, BUFFER_DISTA|NCE=0x%x(%d)bytes\n",
   buf1, buf2, BUFFER_DISTANCE);
   memset(buf2, 'A', BUFSIZE-1);
   buf2[BUFSIZE-1] = '\0';
   printf("Before overflow: buf2 = %s\n", buf2);
   memset(buf1, 'B', (unsigned int) BUFSIZE + OVERSIZE);
   printf("After overflow: buf2 = %s\n", buf2);
}
```

Fig. 5. The code of heap overflow attack

The experimental results show our multi-strategy can easily detect the heap overflow attack. The difference is that only the memory monitoring module can detect the malicious attacks, but not the information flow monitoring.

Information flow tracking does not analyze the security of programs or data coming from external interface. It can defend against malicious attacks on the premise that it needs to mark the external data as untrusted ones when transmitted from any port connected to the CPU. Since the malicious code of the heap overflow attack is written to the SDRAM before the system is running, the information flow tracking policy does not do any tag marking, so it cannot detect the attack.

In the memory monitoring module, it saves function *malloc*'s input parameters (the buffer length) and return value (the buffer start address) in buffer bound information storage unit. Then it monitors the target address of every memory operation. If a target address is among the heap data space, it matches the base address of the target address with the start address saved in the buffer bound information unit until it finds the *buffer*'s length. Then it compares the length with the offset value of the target address. When the length is less than the offset value, a warning is generated. In this case, the length of $buf1$ is less than the offset *BUFFER_DISTANCE+OVERSIZE*, and then an alarm signal is asserted by memory monitoring module.

From the above experimental results, we can see that our multi-strategy, a combination of the information flow monitoring and memory monitoring, can defend various malicious software attacks. Users can properly lower the configuration requirements of the TCR register in information flow tracking to reduce the false positives rate of a single information flow control strategy. Then, the threats missed by information flow tracking can be detected by memory monitoring module. Thus, our proposal formed a security strategy of double protection of embedded systems.

V. Hardware Overhead

To evaluate the performance of our design, we mapped the entire system to an FPGA prototyping platform with a Xilinx Virtex-5 FPGA (XC5VFX70T-FFG1136). We have performed functional verification of the running prototype.

Table 2 shows the comparison of utilization ratios of different types of FPGA resources before and after the integration of our multi-strategy. We can find that the area overhead is almost negligible. For timing overhead, the post-place & route timing report show that our multi-strategy integration has also negligible impact on the critical path delay. The above results indicate that our pro-

Table 2. Research subjects

	Leon3 with security strategy	Leon3
Number of slice registers	10%	10%
Number of slice LUTs	21%	21%
Number used as logic	22%	21%
Number used as memory	1%	1%
Number of route-thrus	1%	1%
Number of occupied slices	40%	39%
Number with an unused flip-flop	57%	57%
Number with an unused LUT	15%	15%
Number of fully used LUT-FF pairs	26%	26%
Number of bonded	37%	37%
Number of block RAM/FIFO	12%	12%
Number of BUFG/BUFGCTRLs	32%	31%
Number of BSCANs	50%	50%
Number of DCM-ADVs	41%	41%
Number of DSP48Es	3%	3%

posed multi-strategy is an ideal solution for embedded processors with very low hardware cost and almost no performance penalty.

VI. Conclusions

We proposed a novel security monitoring strategy of embedded processors in this article. It integrates information flow tracking and memory monitoring into a combined solution, which has been proved to be more effective and can detect more malicious attacks than single solutions. In addition, the hardware cost and the performance penalty of our design are both very small.

In the future work, we will try to integrate more security mechanism based on this work, and will first investigate the main memory data integrity check of embedded system. Memory data integrity check is a very effective way defending against spoofing attacks, relocation attacks and replay attacks[15]. We hope that the method of memory data integrity checking can be also integrated into our security strategy. Next, we will try to achieve the combination of information flow control, memory monitoring and data integrity checking, and build a multi-dimensional security defense system for embedded system.

References

[1] Y. Jin, "Embedded system security in smart consumer electronics", *Proc. of the 4th International Workshop on Trustworthy Embedded Devices*, pp.59–59, 2014.

[2] S. Chen, J. Xu, N. Nakka, *et al.*, "Defeating memory corruption attacks via pointer taintedness detection", *Proc. of the International Conference on Dependable Systems and Networks (DSN)*, pp.378–387, 2005.

[3] M. Ozsoy, D. Ponomarev, N.A. Ghazaleh, *et al.*, "SIFT: Low-complexity energy-efficient information flow tracking on SMT processors", *IEEE Transactions on Computers*, Vol.63, No.2, pp.484–496, 2014.

[4] M.Dalton, H. Kannan and C. Kozyrakis, "Raksha: A flexible information flow architecture for software security", *Proc. of 34th International Symposium on Computer Architecture*, pp.482–493, 2007.

[5] N. Vachharajani, M.J. Bridges, J. Chang, *et al.*, "RIFLE: An architectural framework for user-centric information-flow security", *Proc. of 37th Annual IEEE/ACM International Symposium on Microarchitecture*, pp.243–254, 2004.

[6] V.P. Kemerlis, G. Portokalidis, K. Jee, *et al.*, "Libdft: Practical dynamic data flow tracking for commodity systems", *Proc. of 8th ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments*, pp.121–132, 2012.

[7] G. Venkataramani, I. Doudalis, Y. Solihin, *et al.*, "FlexiTaint: A programmable accelerator for dynamic taint propagation", *Proc. of ACM/IEEE Design Automation Conference*, pp.173–184, 2008.

[8] Z. Liu, X.S. Zhang and X.D. Li, "Proactive vulnerability finding via information flow tracking", *Proc. of the International Conference on Multimedia Information Networking and Security*, pp.481–485, 2010.

[9] M. Dalton, H. Kannan and C. Kozyrakis, "Real-world buffer overflow protection for user and kernel space", *Proc. of the International Conference on Dependable Systems and Networks (DSN)*, pp.395–410, 2008.

[10] C. Cowan, C. Pu, D. Maier, *et al.*, "Stackguard: Automatic adaptive detection and prevention of buffer-overflow attacks", *Proc. of the USENIX Security Symposium*, pp.63–78, 1998.

[11] Zili Shao and Edwin Sha, "Defending embedded systems against buffer overflow via hardware/software", *Proc. of the International Conference on Information Technology: Coding and Computing*, pp.352–361, 2004.

[12] D. Li, Z. Liu and Y. Zhao, "HeapDefender: A mechanism of defending embedded systems against heap overflow via hardware", *Ubiquitous Intelligence and Computing and 9th International Conference on Autonomic and Trusted Computing (UIC/ATC)*, pp.851–856, 2012.

[13] D. Li, Z. Lu, X. Zou, *et al.*, "PUFKEY: A high-security and high-throughput hardware true random number generator for sensor networks", *Sensors*, Vol.15, No.10, pp.26251–26266, 2015.

[14] SPARC Inc, "The SPARC Architecture Manual (Version 8)", *http://www.gaisler.com*, 2016-1-22.

[15] Reouven Elbaz, David Champagne, Catherine Gebotys, *et al.*, "Hardware mechanisms for memory authentication: A survey of existing techniques and engines", *Transactions on Computational Science IV, Lecture Notes in Computer Science*, Vol.5430, pp.1–22, 2009.

LI Dongfang received the Ph.D. degree from Huazhong University of Science and Technology. Currently, he is working at Beijing Institute of Computer Technology and Application. His main research areas include FPGA security and VLSI design. (Email: lidongfang@hust.edu.cn)

ZHAN Xin received the M.S. degree at Huazhong University of Science and Technology. Currently, he is pursuing the Ph.D. degree at Department of Electrical and Computer Engineering, Texas A&M University. His research focuses on embedded system security and EDA. (Email: zhanxin0319@126.com)

TONG Qiaoling received the Ph.D. degree from Huazhong University of Science and Technology. Currently, he is a an associate professor at School of Optical and Electronic Information, Huazhong University of Science and Technology. His main research areas include VLSI design and intelligence computation. (Email: qltong@gmail.com)

ZOU Xuecheng received the Ph.D. degree from Huazhong University of Science and Technology. Currently, he is a professor at School of Optical and Electronic Information, Huazhong University of Science and Technology. His main research areas include VLSI design and the Internet of things. (Email: estxczou@gmail.com)

LIU Zhenglin (corresponding author) received the Ph.D. degree from Huazhong University of Science and Technology. Currently, he is a professor at School of Optical and Electronic Information, Huazhong University of Science and Technology. His main research areas include embedded system security and VLSI design. (Email: liuzhenglin@hust.edu.cn)