LIGHTWEIGHT RECONFIGURATION SECURITY SERVICES FOR AXI-BASED MPSOCS

Pascal Cotret, Guy Gogniat, Jean-Philippe Diguet

Laboratoire Lab-STICC
Université de Bretagne-Sud
Lorient, France
name.surname@univ-ubs.fr

Jérémie Crenne

Laboratoire LIRMM
Université de Montpellier
Montpellier, France
jeremie.crenne@lirmm.fr

ABSTRACT

Nowadays, security is a key constraint in MPSoC development as many critical and secret information can be stored and manipulated within these systems. Addressing the protection issue in an efficient way is challenging as information can leak from many points. However one strategic component of a bus-based MPSoC is the communication architecture as all information that an attacker could try to extract or modify would be visible on the bus. Thus monitoring and controlling communications allows an efficient protection of the whole system. Attacks can be detected and discarded before system corruption. In this work, we propose a lightweight solution to dynamically update hardware firewall enhancements which secure data exchanges in a bus-based MPSoC. It provides a standalone security solution for AXI-based embedded systems where no user intervention is required for security mechanisms update. An FPGA implementation demonstrates an area overhead of around 11% for the adaptive version of the hardware firewall compared to the static one.

1. INTRODUCTION

Embedded systems are facing an increasing number of threats as attackers' motivation is raising every day. Our devices contain many sensitive information (passwords, private information) that needs to be protected from software and hardware attacks. Reconfigurable technologies such as FPGAs can be a good candidate to build trusted devices as they embed processors, memories and application-specific IPs in a single chip with moderate development costs. They also offer several interesting features to build high-performance high-security systems due to their intrinsic performance and adaptive properties. When dealing with logic attacks (e.g. targeting the external memory through code/data corruption) main existing solutions are based on software countermeasures. However, relying the system security on software-only solutions may not be adapted for high constrained embedded systems. We believe adding some hardware mechanisms strongly increases systems security at a very low cost. Monitoring and controlling the communication architecture in a bus-based MPSoC is particularly well adapted to enhance the security of a system as all data are exposed to its structure. Thus if an attacker tries to extract, destroy or modify some data through a logical attack he will have to use the communication architecture. Therefore, enhancing the communication architecture with protection mechanisms is a crucial point. However, when building such a solution several key points need to be addressed: what happens when an attack is detected? How the system should behave? Is it required to increase the protection level of the system? In this paper, we address these questions and propose security update features to provide designers dynamic security levels.
The paper is organized as follows. Section 2 presents related work. Section 3 summarizes a static solution defined in [1]. Section 4 describes our contribution and Sections 5 and 6 propose several results and analysis. Section 7 highlights main perspectives.

2. RELATED WORK

In the literature, several studies have addressed the security of embedded systems [2]. At the communication level, these systems can be protected either by software or hardware mechanisms. Software solutions generally do not require additional hardware but offer low efficiency in terms of latency which can be critical for applications where reactivity is essential to fend off attacks. From an hardware point of view, several solutions have been proposed depending on the communication architecture technology: network-on-chip (NoC) or bus. Regarding NoC-based architectures, Evain et al. [3] propose a solution where security controls are done in each network interface in a distributed manner. A management unit gathers all information from network interfaces according to a security policy.
Fiorin et al. [4][5][6] propose an alternative to this work by adding probes within the interface structure to refine the protection mechanisms. These probes can block incoming traffic according to parameters stored in an embedded memory. A security manager collects information from individual security-enhanced interfaces to detect any collision or error in the traffic. They also provide a adaptive implementation by adding a *shadow memory* (which behaves like a buffer mechanism) to avoid a temporary state of the security enhancement during its reconfiguration.
For bus-based communication architectures, the work of Coburn et al. [7] which is similar to Fiorin's work (without updates) is based on SEI (Security Enforcement Interface) implemented in each interface between an IP and the bus. Each SEI computes information from the data handled by the IP and sends it to a global manager (SEM, Security Enforcement Module) which aims to check SEI data.
This work focuses on an FPGA implementation of a adaptive version of the work presented in [1]. It is inspired by

	SECA [7]	**DPU** [5]	**This work**
Communication standard	AHB/APB	NoC	AXI-4
Adaptive ?	No	Yes	Yes
Threat model	Wide range of soft. attacks	Mainly buffer overflow	Wide range of soft. attacks

Table 1. Comparison with existing works

Fiorin et al. approach [6] with an implementation specific to the AXI bus standard with a low area/latency trade-off. Moreover, several scenarios and a case study are discussed to demonstrate all the features provided in this work.
The key contributions of this work include:

- Demonstration of adaptive firewall enhancements.
- Design of a set of security policies.
- Analysis of attack scenarios and case study implementation.

3. STATIC SECURITY ENHANCEMENTS WITHIN A BUS-BASED MPSOC SYSTEM

This work is based on static security enhancements proposed in [1] and [8]. It provides a low-latency solution based on hardware firewalls embedded in each IP bus interface providing protection against read/write access and format disruptions (Local Firewalls). The firewall connected to the external memory controller (Cryptographic Firewall) adds flexible cryptographic services to protect the memory with confidentiality and/or authentication (Figure 1).

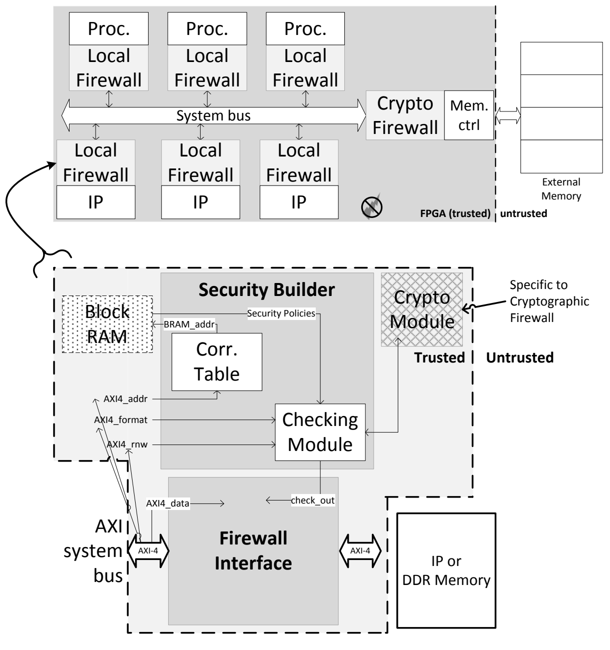


Fig. 1. Firewall structure within a bus-based MPSoC

When a data comes from the AXI system bus, it is stored in the *Firewall Interface* while other information (such as address, format and read/write modes) are sent to the *Security Builder*. The *Correspondence Table* indicates the location of the security policy associated with the bus address: security policies contain cryptographic information (such as keys), read/write access and format rules for a given address space. Then, these parameters are sent to the *Checking Module* which compares the system bus parameters with the values extracted from the security policy; at this step, if cryptographic operations are needed, the *Crypto Module* manages the encryption/decryption and authentication tasks with a dedicated BRAM for cryptographic information storage (this is not the main point of this work). Once *Checking Module* completes its operations, a *check_out* signal is sent to the *Firewall Interface* to confirm or not the data validity (security policies are verified or not). Finally, *Firewall Interface* provides the final data and manages synchronization tasks in order to fit with the output bus interface.
Using this method, the system is protected against logical attacks aiming to tamper the external memory and the external bus without encrypting the whole memory which has a strong impact in terms of latency [1]. Unfortunately, this is a static solution where security policies cannot be changed. Security services update could be done with partial reconfiguration of firewalls or a complete reconfiguration of the FPGA. However these solutions would negatively impact the reaction time of the system in case of an attack and potentially compromise the security. This work focuses on the implementation of a solution based on Block RAM real-time modifications without downloading any new bitstream.

4. SECURITY SERVICES UPDATE REQUIREMENTS

When an attack event is detected, BRAM contents must be updated with new security policies in order to keep a safe execution environment for the target MPSoC. This work proposes an architecture for this purpose as detailed in Figure 2. All the components are connected through an AXI-Lite bus (also known as *Security bus*) and managed by a trustworthy processor (program stored in a trusted ROM) which stores important events in a log file readable by the processor running the main application (timestamps, attack events. . .). Each firewall has a custom bus connection with the monitoring IP. While log timer and monitoring IP are used for detecting and reporting an attack, AXI-Lite BRAM controller connections are used for security policies update.
The first task accomplished in the architecture presented in Figure 2 is the monitoring of attack events by a dedicated IP in the left hand side of the figure. Once an attack is detected, register values are update and an interruption routine is launched for security update on the trustworthy processor depending on the main register value (representative of all attack events). On an attack event, the system security must be updated in order to avoid malicious data leakage. One very important issue during firewalls update is the data availability while switching between two security policies. For this purpose, a mechanism based on the handshake property of the AXI protocol is implemented in each *Firewall Interface* to block outcoming traffic while updating security rules associated with the current firewall.

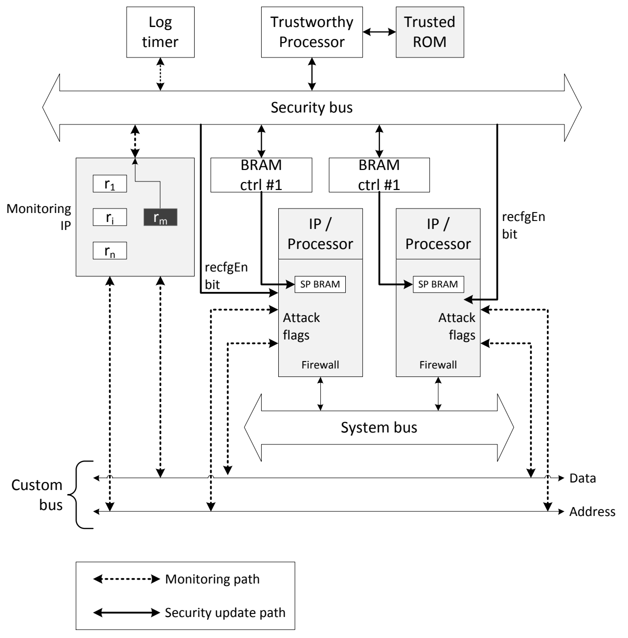


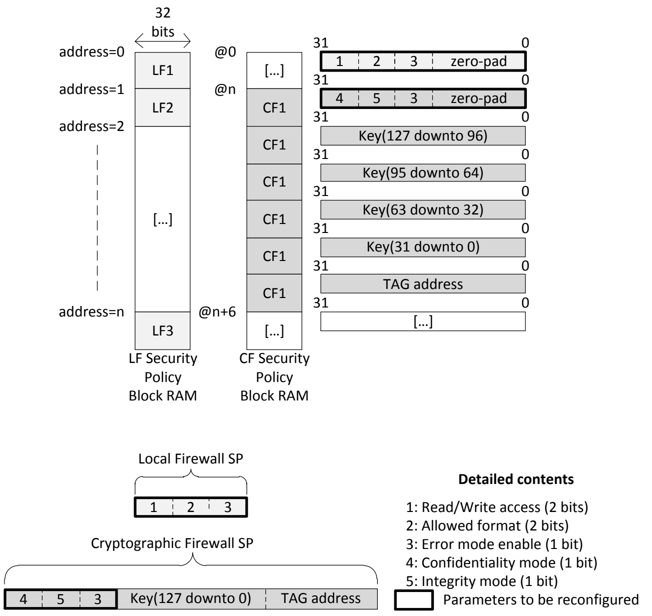
Fig. 2. Architecture of the security/monitoring area

4.1. Hierarchy and evolution of protection modes

Two classes of components are defined based on their ability to manipulate confidential information. Critical IPs (for instance, ciphering algorithms implementation) must not reveal any information when an attack is detected. Extracting keys and/or signatures would be a major threat for the system. In that case as soon as an attack is detected critical IPs are isolated from the system (known as ”error mode” or quarantine). For non-critical IPs, an intermediate protection layer is authorized where reading accesses are still allowed but no writing ones. This feature aims for example to allow a backup of data before IP isolation.
In case of an attack event (detected through interruption routines launched by the monitoring IP), the trustworthy processor saves the current security policy of the attacked firewall in a dedicated on-chip memory and applies a higher security level according to the two schemes previously defined. For instance, if the current protection level of an IP is a ”read-only mode”, the next one could be an ”error mode”, equivalent to a quarantine feature.
In the most critical case (i.e. an attack event is detected even if the IP being monitored is under the ”error mode” protection configuration), it is assumed that a reboot of the complete system is required. Therefore, the initial bitstream and security policies configuration are reloaded. The system restarts from an initial state.

4.2. Switching between the different modes

When an IP firewall must be updated, Security Policies parameters stored in Block RAMs have to be modified (assuming that the whole IP address space is covered by one or more SP, the only component to be updated is the BRAM containing SP values). This work considers only the update of read/write rights (fields 1 in Figure 3). Local Firewall SP is stored on a single 32-bit block while Cryptographic Firewall read/write information is stored in the first 32-bit word of the Cryptographic SP. Writing a Security Policy (stored



32 bits
address=0
address=1
address=2
address=n
@0
@n
@n+6
LF1
LF2
[...]
LF3
[...]
CF1
CF1
CF1
CF1
CF1
CF1
[...]
31
0
1 2 3 zero-pad
4 5 3 zero-pad
Key(127 downto 96)
Key(95 downto 64)
Key(63 downto 32)
Key(31 downto 0)
TAG address
[...]
LF Security Policy Block RAM
CF Security Policy Block RAM
Local Firewall SP
1 2 3
Cryptographic Firewall SP
4 5 3 Key(127 downto 0) TAG address
Detailed contents
1: Read/Write access (2 bits)
2: Allowed format (2 bits)
3: Error mode enable (1 bit)
4: Confidentiality mode (1 bit)
5: Integrity mode (1 bit)
Parameters to be reconfigured

Fig. 3. Security Policy memory layout for Local and Cryptographic Firewalls

in a 32-bit word) in a BRAM takes one clock cycle. Therefore, the update of N Security Policies in one firewall is done in N clock cycles. This time has no real impact on firewall data analysis because data is blocked as soon as an attack is detected (previously described in this work).

5. IMPLEMENTATIONS RESULTS

Implementations were done using Virtex-6 Xilinx FPGA technology (model xc6vlx240t1156-1). This device has around 240,000 logic cells and 15 Mb of Block RAM. First, different implementation options for firewalls are studied.
Then, this work focuses on generic scenarios that serve as a basis for further case study analysis.

5.1. Area

In Table 2, two implementation options are considered: static Local Firewall (based on results of [1]) and the adaptive version.

		Slices	Regs	LUTs	BRAMs
Static solution (Local Firewall)		138	123	293	1
Adapt. version additions	Run-time	6	0	5	0
	Error mode	17	0	18	0
	Misc.	5	13	15	0
	Overhead	+20.29%	+10.57%	+12.97%	+0%

Table 2. Standalone firewalls results

The adaptive version takes into account two features presented in this work: the run-time update and the error mode option. The area overhead due to these enhancements is low (around 11%). The update is based on simple logic elements and the use of bus properties, there is no need to store data in a buffer mechanism while updating security policies of a firewall.

5.2. Latency

Update latency of security policies depends on the number of policies to be updated. The most critical step in this process is the computation of the new security policy configuration done by the trustworthy processor. For a basic implementation of this process, the new configuration is computed in 148 clock cycles.

When more than one attack has to be reported (N firewalls detect an attack), they are first blocked according to their order in the main register of the Monitoring IP. The trustworthy processor computes all the up-to-date security policies. Finally, each firewall is released as soon as its update is completed. Firewalls being updated are not affected because they are not able to transmit any information (handshake signals are controlled by the update logic). This method allows any updated firewall to be used by the main application of the MPSoC system.

The update latency depends on the firewall location in the update queue: the first firewall will be modified in N cycles, while the firewall placed in location $\#k$ in the queue is updated in $k(N)$ cycles because it must wait for the first $k-1$ firewalls to be updated.

6. CASE STUDY

The MPSoC case study considers 2 Microblaze processors, a 64KB shared Block RAM, an image processing IP and an external memory. Each processor has code and data sections in an external memory. This architecture is set with mixed cryptographic options and access rights to get all the options (integrity only, read/write, confidentiality and integrity…). 4 Local Firewalls and 1 Cryptographic Firewall are needed for the protection of this case study. Three options are considered: the MPSoC without firewalls, the static firewall-enhanced MPSoC (based on the results of [1]) and the adaptive version with firewalls. Area results are summarized in Table 3.

	Slices	**Regs**	**LUTs**	**BRAMs**
Unprot. solution	5,446	7,195	8,354	32
Firewalls w/o recfg	7,302 +34.08%	9,848 +36.87%	12,215 +46.22%	51 +37.25%
Adaptive protection	7,442 +36.65%	9,913 +37.78%	12,405 +48.49%	51 +37.25%

Table 3. Table of standalone results

The firewall-enhanced case study has a quite high overhead: this is mainly due to the cryptographic module embedded in the firewall attached to the external memory controller. The logic added for update purposes implies a quite high area overhead of around 40% compared to the static firewall implementation: this is mainly due to the trustworthy processor, the monitoring IP and the security AXI-Lite bus.

7. CONCLUSION AND PERSPECTIVES

In this work, a bus-based MPSoC with adaptive security enhancements (also known as firewalls) is presented. These firewalls protect memories and memory-mapped IPs according to user-defined security policies. By using communication bus properties, it allows developers to get run-time updates with a low area overhead compared to a static solution; furthermore, when security updates are required, there is no invalid data leakage. Mechanisms defined in this work do not need any user event as protection levels are all defined in a dedicated processor.

This work corresponds to a trade-off between SECA [7] and DPU [5] solutions with an implementation on the AXI bus standard implemented in Xilinx tools suite. This work proposes an update feature which is not present in the other bus-based solution. In terms of area overhead over a basic softcore processor, adaptive firewalls have a lower impact (around 11%) than DPU solution (25%, it is based on a *shadow memory* acting as a buffer-like mechanism); SECA solution has the lowest area overhead (6.20%, mainly due to the centralized security manager) but provides a static solution.

Acknowledgment

The work presented in this paper was realized in the frame of the SecReSoC project number ANR-09-SEGI-013, supported by a grant of the French National Research Agency (ANR).

8. REFERENCES

[1] P. Cotret, G. Gogniat, J.-P. Diguet, L. Gaspar, and G. Duc, “Distributed security for communications and memories in a multiprocessor architecture,” in *Proc. IEEE 18th Reconfigurable Architectures Workshop*, May 2011, pp. 326–329.

[2] S. Ravi, A. Raghunathan, P. Kocher, and S. Hattangady, “Security in embedded systems: Design challenges,” *ACM Transactions on Embedded Computing Systems*, vol. 3, no. 3, pp. 461–491, Aug. 2004.

[3] J.-P. Diguet, S. Evain, R. Vaslin, G. Gogniat, and E. Juin, “Noc-centric security of reconfigurable soc,” in *Proc. ACM/IEEE 1st Int. Symposium on Network-on-Chips*, May 2007, pp. 223–232.

[4] L. Fiorin, G. Palermo, S. Lukovic, and C. Silvano, “A data protection unit for noc-based architectures,” in *Proc. IEEE/ACM 5th IEEE/ACM Int. Conference on Hardware/Software Codesign and System Synthesis*, Sept. 2007, pp. 167–172.

[5] L. Fiorin, S. Lukovic, and G. Palermo, “Implementation of a reconfigurable data protection module for noc-based mpsocs,” in *Proc. IEEE 18th IEEE Int. Symposium on Parallel and Distributed Processing*, Apr. 2008, pp. 1–8.

[6] L. Fiorin, G. Palermo, and C. Silvano, “A monitoring system for nocs,” in *Proc. 3rd Int. Workshop on Network on Chip Architectures*, Dec. 2010, pp. 25–30.

[7] J. Coburn, S. Ravi, A. Raghunathan, and S. Chakradhar, “Seca: security-enhanced communication architecture,” in *Proc. 2005 Int. Conference on Compilers, Architectures and Synthesis for Embedded Systems*, Sept. 2005, pp. 78–89.

[8] P. Cotret, J. Crenne, G. Gogniat, and J.-P. Diguet, “Bus-based mpsoc security through communication protection: A latency-efficient alternative,” in *Proc. IEEE 20th Annual Int. IEEE Symposium on Field-Programmable Custom Computing Machines*, Apr. 2012, pp. 200–207.