Bus-based MPSoC security through communication protection: A latency-efficient alternative

Pascal Cotret, Jérémie Crenne, Guy Gogniat, Jean-Philippe Diguet

2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines

Bus-based MPSoC security through communication protection: A latency-efficient alternative

Pascal Cotret*, Jérémie Crenne†, Guy Gogniat*, Jean-Philippe Diguet*
*Laboratoire Lab-STICC, Université de Bretagne-Sud, Lorient (France)
name.surname@univ-ubs.fr
†Laboratoire LIRMM, Université Montpellier 2, Montpellier (France)
jeremie.crenne@lirmm.fr

***Abstract*—Security in MPSoC is gaining an increasing attention since several years. Digital convergence is one of the numerous reasons explaining such a focus on embedded systems as much sensitive and secret data are now stored, manipulated and exchanged in these systems. Most solutions are currently built at the software level; we believe hardware enhancements also play a major role in system protection. One strategic point is the communication layer as all data goes through it. Monitoring and controlling communications enable to fend off attacks before system corruption. In this work, we propose an efficient solution with several hardware enhancements to secure data exchanges in a bus-based MPSoC. Our approach relies on low complexity distributed firewalls connected to all critical IPs of the system. Designers can deploy different security policies (access right, data format, authentication, confidentiality) in order to protect the system in a flexible way. To illustrate the benefit of such a solution, implementations are discussed for different MPSoCs implemented on Xilinx Virtex-6 FPGAs. Results demonstrate a reduction up to 33% in terms of latency overhead compared to existing efforts.**

***Keywords*-communication; security; MPSoC; bus; cryptography; external memory; firewall; latency**

I. Introduction

In daily life as in many industrial environments, electronic devices (smartphones, computers...) manage numerous information that must be protected from potential attackers. These devices generally perform high-performance algorithms in a single chip while manipulating sensitive data (passwords, private information).

For these systems, a trade-off between area, latency and memory consumption is mandatory when building security mechanisms to meet implementation requirements while guaranteeing an efficient protection against attacks. Reconfigurable technologies such as FPGAs can be a good candidate to build such systems as they embed processors, memories and application-specific IPs in a single chip with moderate development costs [1]. A way to reduce the performance impact of security mechanisms is to consider hardware solutions and to develop a distributed protection [2]. One strategic point is the on-chip communication architecture as all data is exposed to its structure. Thus it is of paramount importance to protect these data exchanges in order to keep critical information within the system. Indeed many attacks can be detected through a thorough monitoring of data exchanges within the system. The goal of this work is to propose a solution based on security-enhanced interfaces between IPs in a MPSoC implemented on a FPGA technology.

The paper is organized as follows. Section II presents related work. Section III details the threat model. Section IV describes our contribution and Section V proposes several results. Section VI highlights main perspectives.

II. Related work

In the literature, several studies have addressed the security of embedded systems [3]. At the communication level, these systems can be protected either by software or hardware mechanisms. Software solutions generally do not request 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: network-on-chip (NoC) or bus.

Regarding NoC-based architectures, Evain et al. [4] 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 user-defined security policy. Fiorin et al. [5] propose an alternative to this work by adding probes within the interface structure to refine the protection mechanisms. Each security-enhanced interface, considered as a trusted component, is composed by a set of probes, protection units and a kernel providing network management services. These probes can block incoming traffic according to parameters stored in an embedded context-addressable memory. A security manager collects information from individual security-enhanced interfaces to detect any collision or error in the traffic.

For bus-based communication architectures, one of the most significant work was done by Coburn et al. [6]. This approach is similar to Fiorin's work and 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*). The main limitation of this solution is that all controls are performed in the SEM. It leads to a latency penalty which should be avoided to mitigate security overhead.

Other works provide solutions for the protection of embedded systems: Huffmire et al. [7] introduce the approach of physi-

978-0-7695-4699-5/12 $26.00 © 2012 IEEE
DOI 10.1109/FCCM.2012.42

cal isolation using moats and drawbridges; ARM proposes a commercial architecture name Trustzone [8] which is based on a normal/secure separation using a secure kernel and an access driver. This work is an in-depth description and implementation study of the solution introduced in [2]. It takes advantages of both NoC/bus-based approaches by proposing hardware security enhancements distributed on each IP with low-latency control features. This approach targets bus-based MPSoCs where a limited number of IPs are connected together.

Table I
QUALITATIVE COMPARISON

	SECA [6]	Fiorin [5]	Our work
HW resources	SEI, SEM	Probes, protection units	LF, CF
Security enhancement	Monitoring and verification	Monitoring and verification	Monitoring and verification
Crypto. features	No	No	Yes
Threat model	Wide range of soft. attacks	Mainly buffer overflow	Wide range of soft. attacks
Distributed / Centralized	Centralized	Distributed	Distributed

Our solution aims to be a latency-efficient protection mechanism with an additional cryptographic layer for protection of external memory units (see Table I). Our solution only relies on hardware enhancements (LF and CF as will be explained in Section IV) and does not require modification at the application level neither in the operating system. Compared to existing efforts, we address a wide range of software attacks through a distributed approach and handle cryptographic features to deal with confidentiality and authentication.
The key contributions of this paper include:

- Design and demonstration of dedicated firewalls.
- Promotion of flexible security policy (access right, data format, authentication, confidentiality).
- Development of several applications to analyze security overheads.

III. THREAT MODEL

A. Attack vectors

This work considers that attackers can only tamper with the embedded system using logical attacks (side-channel and other physical threats are not considered). Besides that, it is assumed that the target FPGA is trusted. Therefore, the only way to access the system is through the external memory and the external bus (Figure 1). A solution would be to encrypt and authenticate the whole external memory. Unfortunately, this solution has a high cost in terms of resources consumption and latency overhead [9]. For many applications, building a flexible solution where only the most critical code and/or data sections to be stored in the external memory are protected with cryptographic services is a good choice. Other parts of the memory can be in plaintext or only authenticated [10].
In this case, attackers still have possibilities to compromise the system by tampering non protected parts of the external memory. Indeed, when code and/or data are protected, any change will be detected; but if code and/or data are considered uncritical, no check is performed. Thus, such solutions need to be extended and system designers need other mechanisms to monitor system activity and detect any abnormal behavior. This work proposes to address this point. All communications are checked by security mechanisms mentioned further as firewalls. This additional layer of security provides an efficient solution while maintaining a good area/latency trade-off.

B. Security policies

Security enhancements presented in this work are based on built-in Security Policies (SP) stored in trusted memory units which include a set of parameters required for abnormal behavior detection (see Section IV-D). Basically, it contains read/write access specifications (read/write, write-only, read-only), authorized data format (i.e. 8, 16 and 32 bits) and cryptographic information (confidentiality/authentication modes, encryption key and MAC). Threat model defined in [2] is covered using these parameters. Timestamp tags are used to monitor the access time to the external memory (to detect replay attacks). Spoofing and relocation attacks are also addressed (using address and authentication primitives).

IV. HARDWARE FIREWALLS

In order to prevent any MPSoC system from the threat model previously defined, firewalls are implemented within the system. This section first presents an overview of these blocks. Then a detailed analysis of their architecture from an hardware point of view with latency and memory requirement information is proposed.

A. Overview of the solution

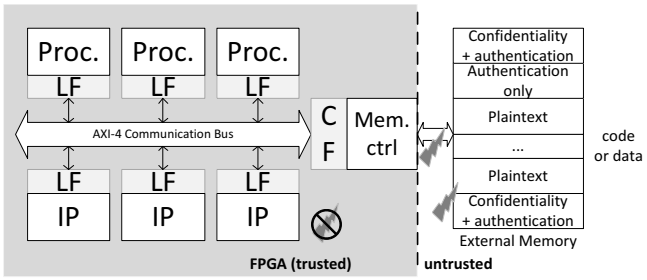

Figure 1. Embedded distributed architecture with security enhancements

The target system is composed of processors, internal memories, dedicated IPs embedded within an FPGA and connected to an external memory. All resources within the FPGA are connected to a bus based on the AXI communication standard from ARM. We propose to add a specific interface to each resource (IP or processor) in order to build a secure gateway to the bus. Using these interfaces (also known as firewalls in this work), we can monitor all

communications before they reach the bus and propagate within the system. Figure 1 shows such a system with internal resources and an external memory. Each resource is connected to a specific interface called Local Firewall providing services for read/write access control and so on. The external memory is also connected to a specific interface called Cryptographic Firewall adding cryptographic features (confidentiality, authentication...). As security parameters can be applied only to specific parts of IPs (defined through security policies), firewalls are also responsible for filtering access to/from the external memory. This feature is of paramount importance when plaintext data or code (which may not have been ciphered) is manipulated. In this case, even processors executing malicious code (that could have been stored in plaintext in the external memory) can be blocked by firewalls. Any illegal read or write access will be detected and discarded.
Security Policies are stored in Block RAMs (memory blocks embedded within the FPGA) which are considered as trusted (no attack is taken into account).

B. Common blocks

Each firewall is composed of several blocks (such as *Security Builder* and *Firewall Interface*) connected to each others (Figure 2). While *Security Builder* is the core managing security policies, *Firewall Interface* is the crossing point with the external world. It acts as a bridge between the system AXI-4 communication bus and the associated IP (custom IP, I/O controller, external memory controller). The whole firewall structure is considered as trusted, the only untrusted area is the external bus and the external memory itself (i.e. everything beyond the external memory controller).

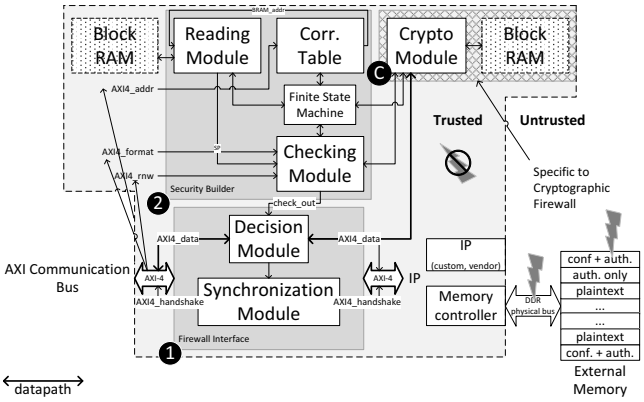

Figure 2. Structure of a firewall

1) Firewall Interface: Firewall Interface is tagged **1** in Figure 2. Two main tasks are performed by a Firewall Interface (FI):

- Once a data is declared as valid, the FI transmits the data to the target element (communication bus or IP), it is performed by the *Decision Module*.
- The FI synchronizes handshake communication signals (such as *AXI_WSTRB*, *AXI_WLAST*, *AXI_WVALID*, *AXI_RREADY*...) in order not to misbehave communication traffic (duplication or omission of data). *Synchronization Module* is based on a set of flip-flops where the clock input signal (i.e. a rising edge) is the acknowledgment signal *check_out* sent by the Security Builder when all the SP-checking operations are completed.

As all these flip-flops are connected in parallel, the latency of Firewall Interface is two clock cycles for a 32-bit data (one for the *Decision Module* and one for the synchronization). For N 32-bit data (without burst or pipeline management), the equation is:

$$latency(N) = N * 2. \quad (1)$$

In case pipeline is activated (not yet considered in this work), the equation for an m-stage pipeline architecture becomes $latency(N) = m + N - 1$. *Firewall Interface* would need 2 cycles for the first data and 1 cycle for each following data (while $data_n$ is processed by the *Synchronization Module*, $data_{n+1}$ is processed by the *Decision Module*).

2) Security Builder: Security Builder is the main component of firewalls (tagged **2** in Figure 2). It is based on four modules:

- **Correspondence Table (CorrTable)**. Security Policies are stored in a trusted Block RAM (left side of Figure 2) and can be identified by an address. *CorrTable* defines the SP address for a given bus target address space in 1 clock cycle. For instance, bus address 0x1234ABCD (in the address space [0x12340000;0x1234FFFF]) is managed by the Security Policy located at BRAM address 0xFF00FF00. The correspondence between these addresses will be performed in the *CorrTable* module.
- **Reading Module (ReadMod)**. *ReadMod* is responsible for reading the Security Policy from the dedicated Block RAM and extracting the security parameters to be sent to the *Checking Module*. For a single 32-bit data, a read buffer is filled in 1 clock cycle then SP parameters extraction (to be sent to the *Checking Module*) requires 1 additional cycle.
- **Checking Module (CheckMod)**. In fact, the *Checking Module* is a set of individual comparators specific to each security parameter (read/write, format...) defined in the Security Policies and computed in parallel (Figure 3). Inverters allow the output to be an overall flag. If

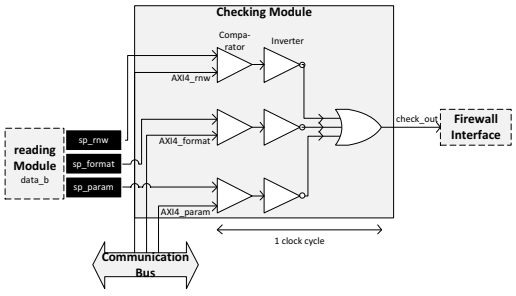

Figure 3. Checking module

inputs of a comparator do not match, output signal goes high. A Security Policy requires 1 clock cycle to be verified.

- **Finite State Machine (FSM)**. *FSM* manages the algorithm running in each firewall.

Timing diagram in Figure 4 represents the behavior of the *Security Builder*. According to the timing diagram, the verification of a single 32-bit data takes 4 clock cycles. Therefore, checking N 32-bit data without pipeline is defined by this equation:

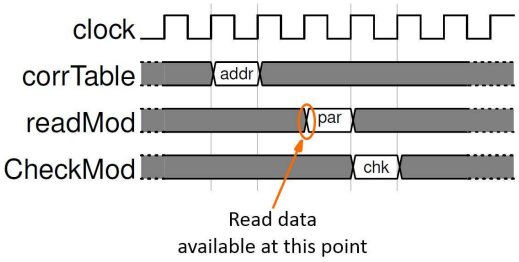

Figure 4. Security Builder timing diagram

$$latency(N) = N * 4 \quad (2)$$

In case pipeline is activated (not yet considered in this work), the equation for an m-stage pipeline architecture becomes $latency(N) = 3N - m(m-2)$. At this stage, the only difference between Local and Cryptographic Firewall is the *Security Builder*. As Security Policies do not contain the same security parameters, *Reading Module* and *Checking Module* have a different structure (see IV-D). Furthermore, another block (trustworthy for cryptographic operations) is implemented in CF.

C. Cryptographic Module

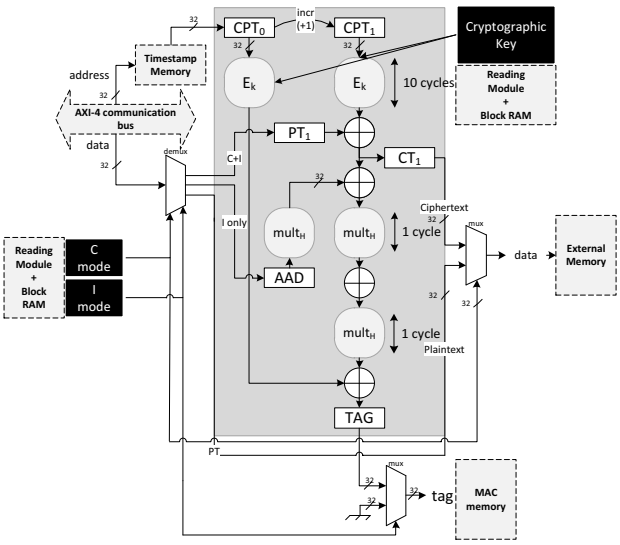

Figure 5. AES-GCM architecture for a 32-bit data in write mode

The solution presented in this work provides flexible cryptographic services based on the AES-GCM algorithm [11]. As shown in Figure 5, a single core has been developed to perform "confidentiality and authentication" or "authentication only" (mux and demux ports are routed according to Security Policy parameters related to confidentiality and authentication modes *Cmode* and *Imode*); keys needed for encryption/decryption are sent by the *Security Builder* (as for MAC information). In Figure 5, encryption of a 32-bit data (E_k uses a 128-bit key and a 128-bit vector containing the 32-bit data padded with zeros), is done in 10 clock cycles and authentication is done in 2 clock cycles (through 2 multipliers $mult_H$ [10]). The overall latency for a set of N 32-bit data protected by confidentiality and authentication is:

$$latency(N) = 10 + (10 + 2) * N \quad (3)$$

Using the AES-GCM algorithm, firewalls can perform low-latency cryptographic features with an acceptable resources consumption. Especially for authentication, AES-GCM takes 2 clock cycles while implementation of hash functions such as MD5 (respectively SHA-2) takes 64 (respectively 80) cycles to do so. The tag issued from the AES-GCM core is not ciphered since it is stored in a trusted embedded memory (Block RAM) separated from the SP memory. As Block RAMs are dual-port memories, one port is left empty for further reconfiguration by a dedicated processor.

D. Security Policies

Each firewall (Local and Cryptographic) has its own 16KB Block RAM containing Security Policies directly connected to the *Reading Module*. Security Policies are indexed by the *Correspondence Table* previously defined. Local Firewall SP is stored on a single 32-bit block while Cryptographic Firewall SP is stored on 6 blocks (LF and CF in Figure 6). Each block is indexed by an address (for instance, address 0) which helps the *Reading Module* to know where to read the first block of the target Security Policy. In case of reading a Cryptographic Firewall Security Policy, a few logic is added to the *Reading Module* in order to read all the SP blocks only knowing the address of the first block.

Figure 6 shows a layout of Block RAMs containing Security Policies for Local and Cryptographic firewalls. As BRAM data ports are 32-bit wide, a Security Policy for a LF is stored

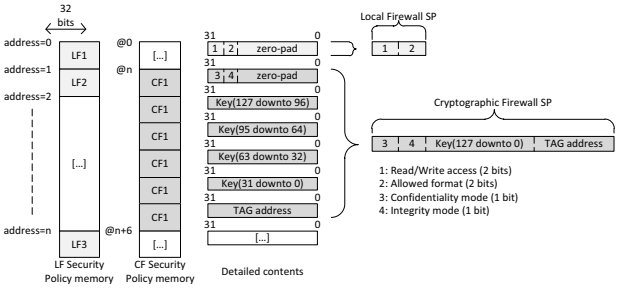

Figure 6. SP memory layout for Local and Cryptographic Firewalls

on 32 bits while CF SP is stored on 6*32 bits. Therefore, in terms of latency, reading a LF Security Policy takes 1 clock cycle (reading a 32-bits word in a block RAM is done in 1 clock cycle) instead of 6 for a SP set for a Cryptographic Firewall. Each firewall has its own Block RAM, separated from Timestamps and MACs memories, with one port left empty for reconfiguration by a specific processor (ability to reconfigure the overall protection if an attack is detected).

The structure of fields labeled as #1, #2, #3, #4 is described in Table II. For confidentiality and authentication, related

Table II
SECURITY POLICY STRUCTURE

		Values			
Field	**Rule**	00	01	10	11
#1	Read all.	No	No	Yes	Yes
	Write all.	No	Yes	No	Yes
#2	Format	4	8	16	32

		Values	
Field	**Rule**	0	1
#3	Confidentiality	No	Yes
#4	Authentication	No	Yes

bits allow to correctly connect data ports to the AES-GCM core in order to perform all the needed cryptographic modes (authentication only, plaintext and so on).

V. RESULTS

All the following results have been implemented on a ML605 Xilinx board including a Virtex-6 xc6vlx240t1156-1 FPGA. This device has around 240,000 logic cells and 15 Mb of Block RAM. First, performances of firewalls are given as standalone blocks. Then, case studies are proposed to verify performances of such security enhancements and compare their efficiency to existing efforts.

A. Standalone results

Area:

Table III
DETAILED SYNTHESIS RESULTS OF FIREWALLS

		Slices	Slice Regs	LUTs	# BRAMs
Local Firewall	**FI**	76	120	68	0
	SB	23	3	55	1
	Total	99	123	293	1
Crypto Firewall	**FI**	76	120	153	0
	SB	23	3	55	1
	CM	1,166	2,038	2,396	14
		89.42%	94.31%	89.10%	93.33%
	Total	1,304	2,161	2,689	15
MicroBlaze		1,179	1,298	1,829	10

All the measures were compared to the Xilinx Microblaze softcore processor (default configuration with 8KB data and instructions caches). Each firewall (Local and Cryptographic) is implemented with two dummy security policies. A Local Firewall has low area consumption while Cryptographic Firewall exceeds Microblaze results. This is mainly due to CM (the AES-GCM core) which consumes around 90% of the CF area (see Table III). Otherwise, firewalls are lightweight in terms of consumed area compared to the Microblaze microprocessor (in terms of registers, a Local Firewall takes around 10% of resources needed for a Microblaze processor).

Once we know the resources needed for each firewall, a set of rules can be defined to estimate the area needed for a multiprocessor architecture with x Local and y Crypto Firewalls. Following equations set takes into account the "worst case" implementation values with 10 SPs for each *Correspondence Table* (this limit is due to the implementation of the *Correspondence Table* which needs 3 registers for each security policy while implementation tools allow 32 configurable registers at most for each firewall).

$$numSlices = 138 * x + 1,304 * y \quad (4)$$

$$numRegs = 123 * x + 2,161 * y \quad (5)$$

$$numLuts = 293 * x + 2,689 * y \quad (6)$$

Using this equations set, a designer is able to compute the area consumed by security enhancements on his own MPSoC for a fixed number of Security Policies embedded in each firewall. It is important to note that the *Correspondence Table* area increase regarding the number of SPs is acceptable: from 9 up to 48 slices and 32 up to 117 regarding LUTs (respectively for 1 and 10 SPs).

Latency:

In order to measure the latency of firewalls, simulations were done to generate the following scenarios:

- S0: Latency of a Local Firewall.
- S1: Communication between two on-chip components.
- S2: Communication between a processor and the external memory in C+A mode.
- S3: Communication between a processor and the external memory in Authentication only mode.
- S4: Communication between a processor and the external memory in plaintext mode.

Each scenario was run for a single 32-bit data: S0 is a reference measure, others scenarios involve two components of a firewall-enhanced MPSoC architecture (S2 to S4 involve one Local and one Crypto Firewall while S1 needs two Local Firewalls). Figure 7 shows the results of these implementations.

Of course, the most critical scenario in terms of latency (28 cycles) is when a data is written in the external memory in S2 (in case of a read, the penalty is not so high). S1 and S4 are quite close because, in this implementation, there is no *Checking Module* (1 cycle saved) in a CF and reading a Cryptographic SP takes 5 more cycles than for Local Firewall: therefore, S4 takes 4 more cycles than S1.

Memory occupancy:

In this section, only Security Policies are considered. Timestamps and Tags (MACs) memories occupancies are not considered as they depend on the number of data required by

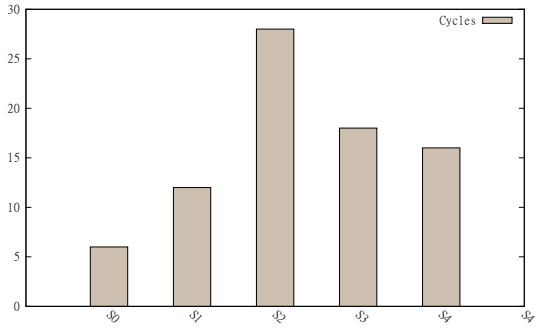

Figure 7. Latency scenarios

the application [9]. According to the formulation of Security Policies, a single policy for a Local Firewall takes 4 Bytes (24 Bytes for a CF because of cryptographic material). Two options are considered in this work:

- Each CPU has uniform rights on other IPs: for instance, 1 CPU can write the whole DDR memory in plaintext mode. This case can be represented by one Security Policy.
- Each CPU has N "options" on each target IP: in this case, for example, 1 processor can read a plaintext address space of the external memory and write a ciphered and authenticated data in another address space (see Figure 1). Several Security Policies are used in this option.

The second option is a generalization of a basic implementation. The more instances of SPs we have for each component of an architecture, the more memory is needed. Figure 8 shows the number of bytes consumed for N IPs (X axis) over M Security Policies. For a fixed number of IPs, memory

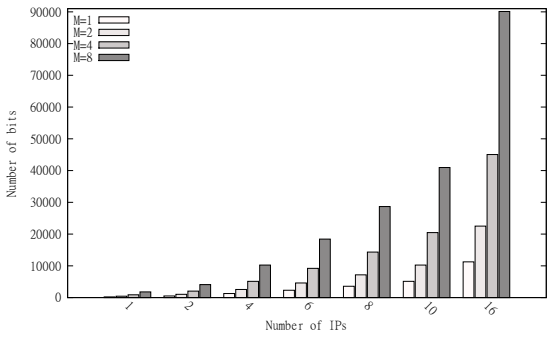

Figure 8. Memory occupancy

occupancy follows a linear function. Considering a large system (16 IPs + 1 external memory, which represents a threshold for a single-layer communication bus), Security Policy consumes around 90,000 bits (nearly 11KB, less than 20% of Block RAMs available on a Xilinx Virtex-6 xc6vlx240t1156-1 FPGA).

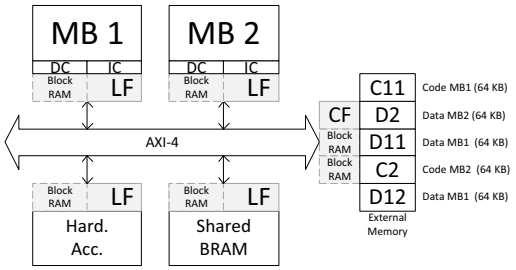

Figure 9. Case study architecture

B. Case study #1: custom architecture

1) Hardware: The case study is based on a dual-processor architecture (Xilinx Microblaze softcores with 8KB data and instruction caches). There are two IPs: "Shared Memory" is a shared Block RAM of 64 KB and Hard. Acc. is an image processing IP (threshold function) with a set of registers (Figure 9). The external memory contains data and code sections for both processors. This work considers that external memory is split in five parts (64KBytes each): two parts for processor #1 (code and data) and three sections for processor #2 (code and two data sections, with different cryptographic policies).
Contents can be either protected by confidentiality and authentication (C11 and D11), authentication only (D12) or even plaintext (C2 and D2). Each processor also has specific access to the two internal IPs (in terms of Read/Write access context): for Hard. Acc., MB_1 can read and write while MB_2 can only write; for Shared BRAM, MB_1 can only read, MB_2 read and write.

2) Benchmark application: In order to set up this system as a base for further analysis, an application was defined to make all key situations happen (accesses to the external memory, internal communications and so on). This benchmark application, represented by the sequence diagram in Figure 10, is based on a symbolic JPEG file (previously ciphered and written in the external memory by a configuration processor) which is processed and transmitted from one block to another. According to the former hardware description, the external memory contents are split in different sections with several Security Policy instances.

Table IV
SYNTHESIS RESULTS OF THE CASE STUDY #1

	Slices	Slice Regs	LUTs	# BRAMs
Case study w/o firewalls	5,446	7,195	8,354	32
Case study w/ firewalls	7,302 +34.08%	9,848 +36.87%	12,215 +46.22%	51 +37.25%

When the case study is implemented with security enhancements, the area consumption is around 40% higher than the unprotected version (nearly 46% for LUTs in Table IV).

Table V shows latency overheads for a couple of applications:

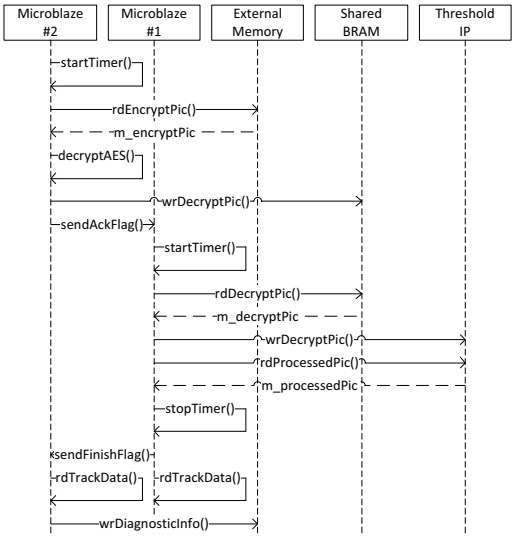

Figure 10. Benchmark sequence diagram

- *picProc* is the application described in the sequence diagram.
- *picDrm* is a sample DRM application run by one processor while the other processor does read/write operations.
- *picDec* is a classic software AES decryption done by one processor which also accesses the external memory.

Table V
BENCH APPLICATIONS RESULTS

	Exe. time (ms)	**# of cache miss**	**# of DDR access**	**Lat. Overhead**
picProc	3,623	12,672,395	34,063,398	17.76%
picDrm	1,084	3,017,237	9,462,055	9.43%
picDec	382	793,226	4,736,966	4.18%

picProc is the largest application (in terms of code size): as there are more accesses to the external memory, the penalty induced by firewalls is higher.

C. Case study #2: other platforms

We also evaluated our solution for two platforms. The first one is the NEC's MP21x application SoC, also studied by Coburn et al. [6]. Figure 11 shows a simplified view of this system. There are 3 ARM 926 CPUs, a cryptoprocessor able to run several algorithms (RSA, AES and so on). NEC MP21x platform also has Security Policies (as it has been defined in previous sections). For this case study, it is considered that each CPU has uniform rights on other IPs (for instance, 1 CPU has only one set of rights on the cryptoprocessor). The second platform is the Samsung Exynos 4210 SoC based on a dual-core ARM Cortex A9 platform used, for instance, in smartphones such as the Galaxy SII (from Samsung). It also has secure memory blocks, video peripherals and a cryptographic engine.

1) Area: According to the NEC platform schematic, we consider that SDRAM and Flash are critical memory units

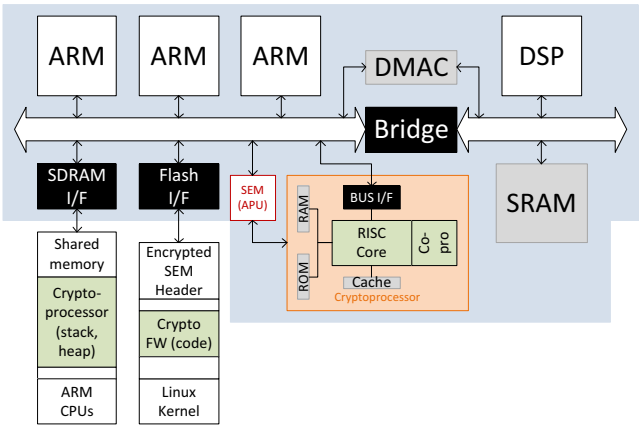

Figure 11. NEC MP21x simplified architecture

and need to be protected by cryptographic services. Therefore, 5 LF and 2 CF (SDRAM et Flash I/F) are needed for this case study. For the Exynos 4210 platform, 12 LF and 1 CF (Memory Interface) are required.

Table VI
SYNTHESIS RESULTS OF NEC AND SAMSUNG PLATFORMS

	# of LF + CF	Slices	Slice Regs	LUTs	# BRAMs
MP21x	5 + 2	3,298	4,937	6,843	35
Exynos 4210	12 + 1	2,960	3,637	6,205	27

According to Table VI, firewall enhancements takes 8.75% of a medium-scale Virtex-6 FPGA (xc6vlx240t) for the NEC MP21x platform. A Samsung Exynos 4210 platform is secured by firewalls with 7.86% of a medium-scale Virtex-6 FPGA (xc6vlx240t).

2) Memory occupancy: We consider that in a MPSoC system, each IP needs only 1 Security Policy. To calculate the equation giving the memory occupancy for x LF and y CF, it is assumed that:

- Each IP attached to a LF has access to all other IPs of the system (including external memories). There are $x + y$ IPs in the system, an IP cannot interact on itself and each LF security policy takes 32 bits. Therefore, total memory occupancy of LFs follows the equation $(x + y - 1) * 32x$.
- Each IP attached to a LF has access to the external memory (in this case, each IP needs 1 Security Policy). In the external memory mapping, there is an address space for each IP, each address space is managed by a 192-bits CF security policy. Memory occupancy of a Cryptographic Firewall is governed by the equation $192x$ ($192xy$ if y CF in the system).

The evolution of memory occupancy for NEC and Samsung platforms follows the surface equation $32x^2 + 224xy - 32x$. Even with a high protection granularity (20 Security Policies for each IP), SP memory occupancy costs are quite acceptable

(864 Bytes for Samsung Exynos 4210 platform and 360 Bytes for NEC MP21x). In this analysis, tags storage is not considered and this point can significantly increase the memory overhead if a uniform security policy is considered [9].

D. Comparison over existing works

A comparison with related works is done in terms of area overheads. The area results for our work does not take into account the Cryptographic Module because Coburn [6] and Fiorin [5] do not have ciphering features in their solutions. Xilinx Microblaze processor was taken as a reference for

Table VII
SECURITY OVER PROCESSOR AREA RATIOS

	$\frac{area(security\ enhancement)}{area(processor)}$
Coburn [6] (16KB / 150 MHz)	6.20%
Fiorin [5] (8KB / 100 MHz)	25%
Our solution (8KB / 100 MHz)	11.30%

Table VII, individual configurations are indicated between brackets (caches size and clock frequency). In Table VII, our solution has a ratio lower than the *Data Protection Unit* system [5]: this may be due to the CAM system which is more complex than a Block RAM. SECA-based solution has a lower impact because the kernel (containing security configurations) is not included in this estimation while security policies stored in firewall BRAMs are taken into account in this analysis.

In order to compare our solution to a centralized approach like SECA where controls are done in the manager entity, it is assumed that each transfer requires 4 more cycles (a normal transfer on AXI bus takes 2 clock cycles [12]): it represents the latency to communicate between a firewall and the security manager of the system. Therefore, giving the *picDec* application defined in Section V-B, a centralized implementation like SECA would give a 6.27% latency overhead while our solution has a 4.18% overhead: it represents a 33% latency decrease.

VI. CONCLUSION AND PERSPECTIVES

This work proposes distributed and flexible security enhancements for bus-based MPSoC that embed protection mechanisms where existing solutions are mainly centralized (which involves additional latency overhead for communication between security elements). Our solution has low latency overheads and can be further improved by pipelining data management. In terms of area, LF and CF have low resources consumption: the major penalty is due to the AES-GCM core. Our solution is a layer above the communication protocol as our security enhancements do not modify the communication protocol between the processors and the other parts of the system. We plan to integrate reconfiguration of security services (i.e. modification of security policies) to counter some attacks against the system: this could be done quite easily because Block RAMs containing Security Policies have one data port left for a reconfiguration processor (it raises other issues such as firewall behavior during reconfiguration of SPs). In this work, policies are defined using address domains, it can be interesting to study the adaptation to thread-specific security where each thread has its own security policy. Finally, there is the question of scalability: a single-layer bus allows around 32 IPs; we can imagine to study benefits of firewall implementations on a multi-layer bus or a network-on-chip.

REFERENCES

[1] B. Badrignans, J.-L. Danger, V. Fischer, G. Gogniat, and L. Torres, "Security trends for fpgas," in *From Secured to Secure Reconfigurable Systems*. Springer, 2011, p. 252.

[2] P. Cotret, J. Crenne, G. Gogniat, J.-P. Diguet, L. Gaspar, and G. Duc, "Distributed security for communications and memories in a multiprocessor architecture," in *Proc. 2011 Reconfigurable Architectures workshop (RAW)*, 2011, pp. 326–329.

[3] 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, 2004.

[4] 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*, 2007, pp. 223–232.

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

[6] 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*, 2005, pp. 78–89.

[7] T. Huffmire, B. Brotherton, G. Wang, T. Sherwood, R. Kastner, T. Levin, T. Nguyen, and C. Irvine, "Moats and drawbridges: An isolation primitive for reconfigurable hardware based systems," in *Proc. Int. Symposium on Security and Privacy*, 2007, pp. 281–295.

[8] X. Yan-Ling, P. Wei, and X.-G. Zhang, "Design and implementation of secure embedded systems based on trustzone," in *Proc. 2008 Int. Conference on Embedded Software and Systems*, 2008, pp. 136–141.

[9] R. Vaslin, G. Gogniat, J.-P. Diguet, R. Tessier, D. Unnikrishnan, and K. Gaj, "Memory security management for reconfigurable embedded systems," in *Proc. 2008 Int. Conference on Field-Programmabkle Technology*, 2011, pp. 153–160.

[10] J. Crenne, R. Vaslin, G. Gogniat, J.-P. Diguet, R. Tessier, and D. Unnikrishnan, "Configurable memory security in embedded systems," *ACM Transactions on Embedded Computing Systems*, 2011.

[11] D. A. McGrew and J. Viega, "The security and performance of the galois/counter mode (gcm) of operation," in *INDOCRYPT*, 2004, pp. 343–355.

[12] N.-C. Chang, Y.-Z. Liao, and T.-S. Chang, "Analysis of shared-link AXI," *IET Computers & Digital Techniques*, vol. 3, no. 4, p. 373, 2009.