

A Bio-Inspired Framework for Secure System on Chip

Ayan Mandal, Suman K. Mandal, Aalap Tripathy, Nikhil Gupta and Rabi Mahapatra
Department of Computer Science & Engineering
Texas A&M University
{ayan,skmandal,aalap,nkgupta,rabi}@cse.tamu.edu

ABSTRACT

In this paper we propose a bio-inspired security framework for Network-on-Chip based many-core system-on-chip (SoC). The proposed framework is intended to provide an infrastructure suitable for handling security at chip level during potential system level attack. A hardware security core is introduced on-chip to manage SoC security challenges using a dedicated communication network. A signal communication model has also been proposed to develop various safety related protocols to address some of the safety concerns. The proposed framework is capable of detecting and isolating a compromised core from the rest of the system on chip so as to prevent it from further exploitation. A simulation based experimental set up has been proposed to demonstrate the security infrastructure. It has been shown that the program execution overhead due to build in security infrastructure ranges 3-6 %. Finally the area and power overhead due to the security core is also given.

Categories and Subject Descriptors

Novel SoC Architectures [Fabrics/Network on Chip]

General Terms

Design, Security

Keywords

Network on Chip (NoC), Security, Design, Multiprocessor System-on-Chip (MP-SoC), Embedded System, Artificial Immune System

1. INTRODUCTION

Network on Chip [1] enables the integration of many processing units in a System on Chip by providing efficient communication architecture. The advantages are availability of high bandwidth for data communication and no requirement for centralized synchronization. But this comes at the cost of latency and security. Networked embedded systems are vulnerable to numerous security attacks [2]. A typical many-core embedded system consists of various cores (processors, memory, ASIC, FPGA, I/O controllers etc) connected with a communication medium. In this paper, we propose a framework for secured System on Chip (SoC) inspired by cellular immune response mechanism.

Cellular immune response system involves three actions: anomaly detection in a cell, signaling to other cells, and corrective measures to prevent further damage to other cells. A similar approach can be applied for SoC security. A dedicated security core can perform the actions necessary for security management in SoC thus acting like an immunity center. In biological immune system, proteins and immune cells perform the task of signaling between cells and the immune system. In a SoC the secured signaling can be modeled using special communication packets on dedicated communication channel (Figure 1). In this work, we are proposing a suitable SoC security framework based on the biological principles of immunity capable of detecting and isolating a compromised core from the rest of the system on chip

so as to prevent it from further exploitation. A simulation based experimental set up has been implemented to demonstrate the validation of the security infrastructure. The contribution of this work can be summarized as:

- Modeling of SoC security mechanism based on Biological Immune System
- Design of a Hardware Security Core (SC) to manage SoC security using the proposed model.

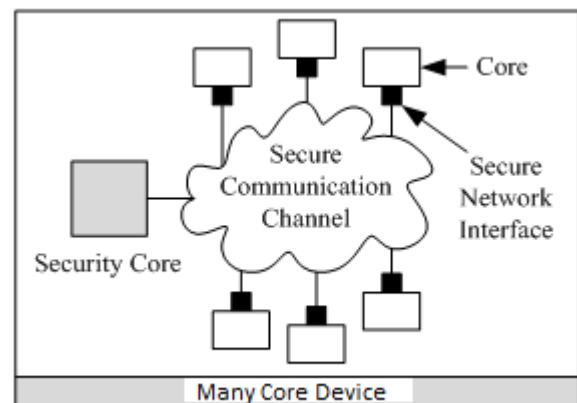


Figure 1: A Many-Core System with Security Core and Dedicated Secure Channel to Cores

- Design and Implementation of a Network on Chip infrastructure for SoC interconnects to support the proposed security model.

The rest of the paper is organized as follows. Section 2 discusses the related works in the literature. The Bio-Inspired Security Model has been described in Section 3. Section 4 contains the Design and Implementation of the Security Core and the Network on Chip infrastructure. In Section 5 the evaluation is presented followed by Conclusion in Section 6.

2. RELATED WORK

Design of security-aware communication architectures has been introduced as SECA [3] which relies on both a single Security Enforcement Module (SEM) and a Security Enforcement Interface (SEI) for each slave device connected to the bus. They use AMBA bus which can implement features such as (a) address-based protection, (b) data-based protection and (c) sequence-based protection. While NoC has been subject of several studies and evaluations for many-core SoC, security aspects related to NoC implementations has been only recently addressed. In [4] authors present a framework to secure the exchange of cryptographic keys within NoC. Authors in [5] proposed a first solution to secure a reconfigurable SoC based on NoC. The system is composed of Secure Network Interfaces (SNIs) and a Secure Configuration Manager (SCM). The SNIs act as filter for the network and as monitor for the incoming traffic, while the SCM configures

system resources and network interfaces. There have been hardware and software solutions for detecting malicious applications in NoC framework.

[12], fault diagnosis [12], decision support system [14], robust scheduling [15] and multi-optimization problems [16]. All the applications mentioned above uses pattern-matching and learning

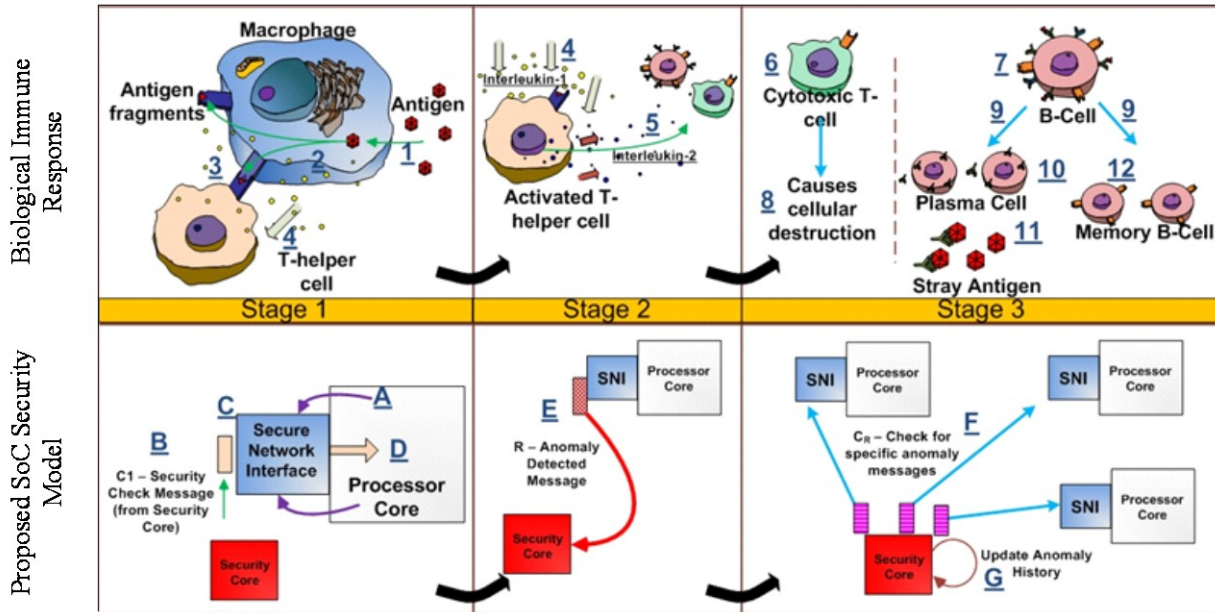


Figure 2: Bio Inspired Framework for SoC Security

In this paper we propose and implement a framework that is capable of detecting and isolating a compromised core from the rest of the system on chip so as to prevent it from further exploitation. This involves design and implementation of a secured communication channel and protocol following the biological immune system signaling. This work is different to the existing work in [4,5] as it provides a suitable prototype for security aware SoC.

3. BIO INSPIRED FRAMEWORK FOR SOC SECURITY

Figure 2 demonstrates an overview of the proposed SoC security infrastructure that is derived from biological immune response mechanism. Activation of an immune response in biological systems is mediated by white blood cells (macrophages). A macrophage’s primary role is to present antigen fragments on the cell surface (actions 1, 2 in Figure 2). Another specialized white blood cell (T-helper cell) interacts with the macrophage to recognize (action 3 in Figure 2) the presented antigen fragment and gets “activated” (action 4 in Figure 2). An “activated” T-helper cell causes the proliferation (action 5 in Figure 2) of cell destroyers (cytotoxic T-cells) and antibody producers (B-Cells) (actions 6 and 7 in Figure 2). Activation of cell destroyers implies a go-ahead for an infected cell’s destruction (action 8 in Figure 2). Antibody producers retain a long term memory of the antigen. This provides “acquired immunity” to secondary infection. The biological immune system (BIS) models have been used in computer security [12, 13], anomaly detection in time series data

mechanisms. BIS being highly parallel and distributed suggests the modeling of an integrated architecture as a multi-agent system (MAS), where separate functions are carried out by individual agents [13]. Moreover the general immune system consists of adaptive processes at local level with useful behavior emerging at global level [12]. In the proposed model these two tasks are assigned to special components such as Secure Network Interface (SNI) and Security Core (SC) respectively. A SNI’s primary role is to identify the current behavior of the core (action A in lower half of the Figure 2). A SNI interacts with “Security Check” message (sent to it from the SC) and compares it with the observed behavior of the core (actions B and C in Figure 2). This message contains the expected behavior of the core as determined by the SC. An anomaly is identified when the observed behavior and the expected behavior do not match. This causes generation of an “Anomaly detected” message which is sent to the security core (action E in Figure 2). The security core responds to an “Anomaly detected” message by sending a “Check for specific anomaly” broadcast to all the cores within the many-core device (action F in Figure 2). Along with this it isolates the compromised core so as to prevent the system from any further damages. The security core will retain a “genetic” memory of the anomalies detected. This will enable it to autonomously produce “Check for specific anomaly” messages in the future. We implement the above scheme by defining the security message classes and the functional model for monitoring core level security as given in Table 1.

Table 1: Message Classes for Secure System on Chip

Class	Type	From	Destination	Content	Remarks
I	Security Check (C1)	Security Core	Individual SNI	Expected statistical behavior for an individual core (may specify expected tolerances)	Periodically generated
II	Behavioral Information (C2)	All SNIs	Security Core	Observed behavior of the system. Intended to inform the security core of the core's current status.	Information from other cores in the message path may be piggy-backed.
III	Anomaly detected (R)	Individual SNI	Security Core	Information regarding the exact anomaly detected.	Priority routing back to the security core
IV	Check for specific anomaly (C _R)	Security Core	All SNIs	Information about a specific anomaly to check for.	Priority routing to the SNIs

3.1 Message classes

Communication between the security core and SNI's is accomplished using four classes of message as detailed in Table 1.

3.2 Monitoring Function Model

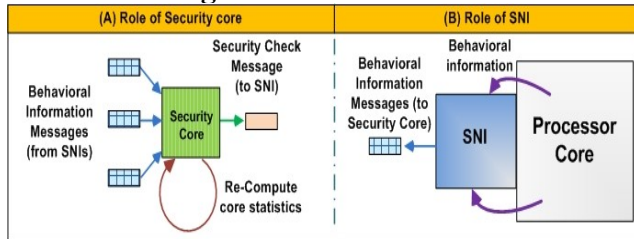


Figure 3: Monitoring Function of Security Core and SNI

The monitoring of core-level security will be carried out by the SC and the secure network interfaces as illustrated in Figure 3. The SNIs will gather various core level statistics and periodically send them to the SC as Behavioral Messages. The Security Core will compute averaged expected behavior of all cores using such Behavioral Messages sent from the various SNI's. It will periodically generate Security Check messages to test individual cores.

3.3 Protective Function Model

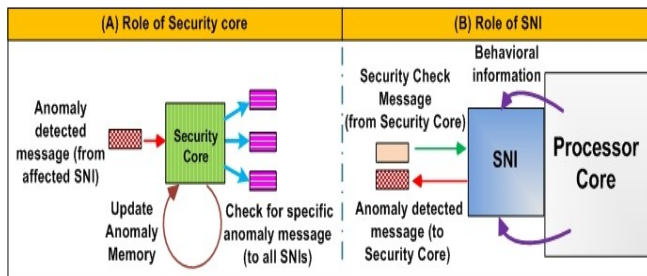


Figure 4: Protective Role of Security Core and SNI

SNI will compare the currently observed behavior of a core to the expected behavior information contained in the security check message. If an anomaly is detected, (core behavior outside specified limits), an "anomaly detected" message will be sent to the SC. An "okay" message need not be produced. A core which does not produce an "anomaly detected" message will be assumed to be working fine. This process is illustrated in Figure 4(A). If an "anomaly detected" message is received, the security core will produce "Check for specific anomaly" message to initiate specific

diagnostic function in the SNIs of other cores. This process is illustrated in Figure 4(B).

3.4 Attack Model:

Attacks in many-core devices may be classified into several categories and are described in Table-2.

Table 2: Attack Types in System on Chip

Attack	Description
Hijacking	Write access in the secure area in order to modify the behavior or the Configuration of the system, it also includes buffer overflow and internal registers reconfiguration.
Extraction of secret information	This second aspect consists in read accesses to data stored in secure targets. The stolen information can be sensitive data (e.g. encryption keys), instructions from critical programs, IP configuration registers and so on.
Reverse Engineering	Theft of intellectual property information through unauthorized reads in memories to obtain pieces of firmware. An example is differential power analysis (DPA) to proceed cipher key extraction.
Denial of Service	This kind of attacks aims to bring down the system performances. The network over utilization downgrades the operability of the system.

Security enforcing mechanisms like strong encryption is inappropriate for SoC due to resource constraints. In this model, we assume that any type of adversarial attack on a processor core will be manifested in the form of some anomalous behavior. The anomaly could be in terms of one or more of the following as envisioned in our approach:

1. Memory Access Pattern – Address ranges, frequency
2. Data Access/Network/Resource Request rate
3. Power consumption
4. Validity of communication destination

4. ARCHITECTURE & OPERATION

4.1 Secure Network Interface

4.1.1 The State flow diagram

Figure 5 shows the operating protocol of the security module inside SNI. It starts in the "Idle" state. Whenever it detects any activity in the SNI indicating the start of a new application, it

switches to “Init” state. There it decodes the application identifier and sends a request to the security core for an expected behavior. Then it switches back to “Idle” state. When the security core sends C1 message (Expected Behavior), it populates its expected behavior entry. Getting the expected behavior from the security core improves the memory requirement. For subsequent C1 messages from the security core, it compares the difference with the expected behavior and sends the digest back to the security core. When it receives a Cr message (Check for specific anomaly), it updates the security probes with new virus signatures and also verifies if the anomalous behavior remains persistent in subsequent phases. It sends the report by going to “Transmit” state. Eventually when the anomaly is confirmed by the security core, it goes into “Reconfigure” state. In this state, it stops all the transactions to and from the core. It also marks the application vulnerable by informing the security core and sends corresponding signatures.

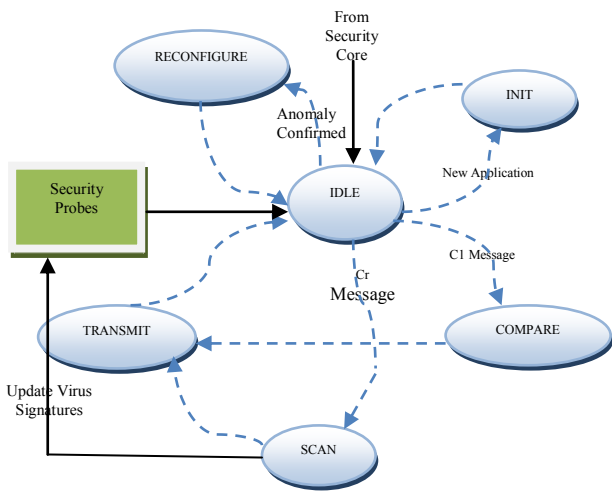


Figure 5: State Flow Diagram of Security Module

4.1.2 Architecture

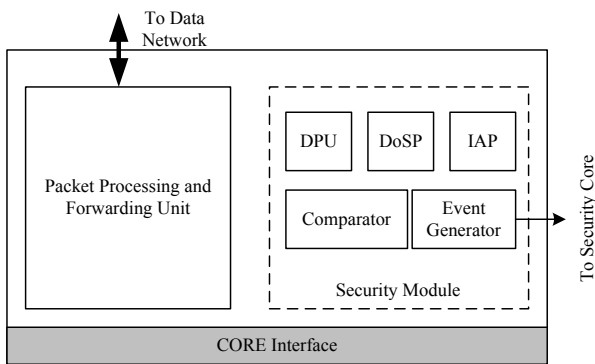


Figure 6: Secured Network Interface

The SNI enables the analysis of the traffic as it is inserted by the core which reduces the cost for implementing probes necessary for threat detection. We have used the hardware probes suggested by [7]. As shown in Figure 6 consists of a Data Protection Unit (DPU), Illegal Access Probe (IAP) and Denial of Service Probe (DoSP). DPU is a hardware block which has the knowledge regarding the access to a given memory region. This is updated

during the configuration phase by the security core. Each time a new application runs on the system it needs to be configured by the security core. It can also store the configurations of the previously run applications. IAP detects attempts to illegally access restricted memory blocks or range of addresses in shared memory systems. DoSP detects Bandwidth reduction or draining attacks. So during an evaluation phase, the SNI collects all the data from the network traffic. Other than this it consists of a comparator block and one Event Generator. SNI constructs the behavior of a certain application during the period for example the communication pattern with other cores, the latency, any memory violation or security threat detection. Next, the comparator is used to compare against the expected behavior sent from the security core. We have used a bloom filter [10] approach for comparison of behavioral statistics. This approach is efficient in terms of memory and hardware overheads. This operation results in to detection of security violation. An event is generated to notify the security core of any security violations. An Event Generator creates a security message as in [11] to represent an event which composes of an Identifier, a Timestamp, a Producer and several Attributes. Identifier defines the type of attack from the Producer node at the Timestamp. The attributes are used to pass on the statistics regarding the core behavior which is used by the security core to update the expected behavior during the next evaluation phase.

4.2 Security Core

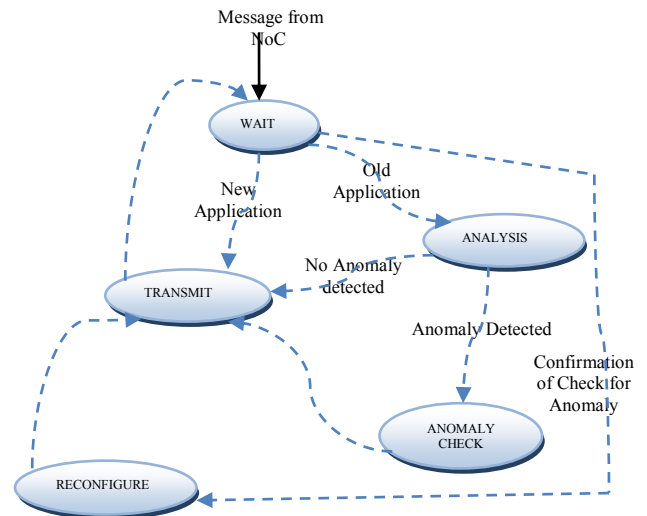


Figure 7: State flow diagram of Security Core

4.2.1 The state flow diagram

Figure 7 shows the state flow diagram of the security core. It starts in an “Idle” state. It maintains a communication profile of applications running on the cores. This is configured during the boot up and is inaccessible from outside SoC. However for the unknown applications, the security core goes through a training phase to calculate the transactional behaviors. In “Idle State”, if it receives activity information of a new application in a core, it makes a copy of the expected behavior and switches to “Transmit State” thus sending a “Security Check” message.

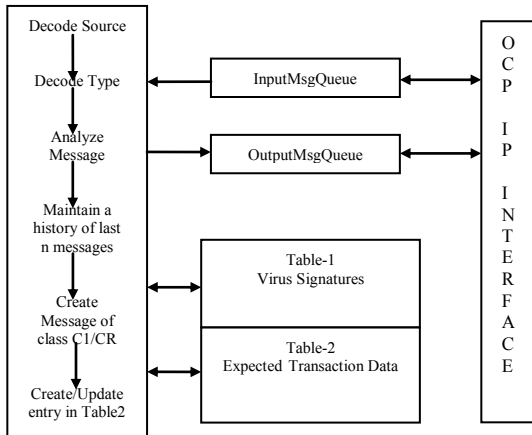


Figure 8: Security Core Architecture

The subsequent messages from the same application change the state to “Analysis State”. In this state, the security core maintains a profile of the transactional behavior of the core. Accordingly it updates the expected behavior table and regenerates the “Security Check” message and moves to “Transmit phase”. When an incoming message indicates the presence of any violation, it switches to “Anomaly Check State”. In this phase the task is to confirm the attack and also search for similar vulnerabilities in other cores. This is achieved by “Check for specific Anomaly”. This also involves scanning for specific search strings so as to identify tainted data in the core and isolate them. Then it switches to “Idle State”. When there is confirmation of attack in response of the “Check for specific Anomaly” message, it switches to “Reconfiguration State”. The security core immediately reconfigures all the SNIs so as to isolate the compromised core. It also updates its own expected behavior table to blacklist the given application. It also adds an entry to the virus signature patterns as it is more likely that the attack will come back again.

4.2.2 The Architecture

Figure 8 shows the architecture of the security core. It communicates with a CNI by OCP protocol. Making the core OCP compliant enables its seamless integration as an IP core in any Network on chip following OCP-IP protocol for communication. It accepts request from the OCP-IP interface as a slave and transmits the configuration messages to the other cores by a master interface to OCP. Thus it maintains two different queues to communicate with OCP-IP interface. Along with this it maintains two tables in memory to store the expected behavior of all applications running in the system and a subset of virus signatures. As the security core receives a message from OCP-IP, it finds out the message class and the source. For a new application, it creates the C1 Message (Expected Behavior) from the table saved in the memory and sends it back to the core. For old applications, it analyzes the digest coming from the core and compares with the expected behavior in the table. It then maintains a history of last n (determined by available memory) digests. If the type of the message is “Anomaly detected”, it creates “Anomaly check message”. Now this can be scanning for known signatures or verifying the core behaviors running similar applications. If the message type is “Anomaly confirmed”, it

immediately sends command to corresponding SNI to block all transactions in the core and asks for signatures from the core so as to immune the system from similar attacks in future. This updates the virus signature table as it deletes the previous entries and maintains a list of recent vulnerabilities.

5. EXPERIMENTAL SETUP

We use NoCBench; a System on Chip simulation platform built using SystemC to implement and evaluate the proposed security framework. The platform allows us to simulate full SoC with software and hardware cores modeled in C/C++ and SystemC respectively. It also provides functionality to perform detail simulation of one or more Network on Chip that connects the cores. We have considered a 4x4 mesh topology as shown in Figure 9. Four sparcv8 cores are connected to top four routers through SNI. There are two memory cores connected two bottom two routers. The security core (SC) is connected as shown in the figure. Rest of the routers is provided with dummy cores which send random traffic at a desired flit injection rate. In the following experiment we use a flit injection rate of 25%.

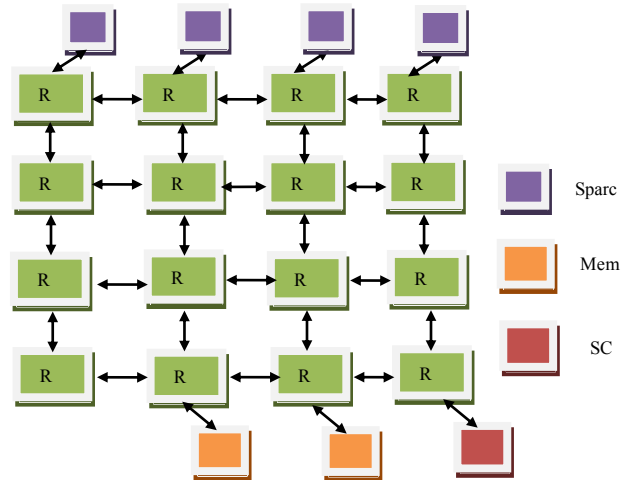


Figure 9: SoC Configuration

5.1 Performance Evaluation

The SNI and Security Core are being implemented using SystemC as hardware modules. They are plugged into NoCBench platform along with other processing cores to model the proposed secure SOC architecture. This allows us to do timing accurate functional simulation of the framework. We have used Mibench [17] as application benchmarks. The following metrics are available for analyzing the performance: 1) Application Run Time Overhead (Completion time of the application in presence of the security core). 2) Effect on Application Throughput (The minimum period at which the input processing core injects data). Mibench has six suites with each suite targeting a specific area of the embedded market. The six categories are Automotive and Industrial Control, Consumer Devices, Office Automation, Networking, Security, and Telecommunications. These categories offer different program characteristics that enable researchers in architecture and compilers to examine their designs more effectively for a particular market segment. So we run a given application benchmark on the sparcv8 cores. In the “Training Phase”, we save the expected behavior of this application by recording the

transactional behavior with other communicating cores. During the “Detection Phase” we run the same application in all the four sparvc8 cores. However one of the applications is instrumented so as to send random read requests. As a result the particular application shows anomalous communication pattern. The security probes in the SNI are configured so as to detect any illegal accesses.

Table 3: Security Core (SC) Metrics

Application	Total Number of cycles(Millions)	
	Original	With SC
Qsort	27.77	28.81
Gsm encoder	32.80	33.37
Gsm decoder	16.21	16.89
Stringsearch	0.77	0.82

The increase in number of cycles for execution thus turns out to be varying from only 3-6%. This is expected from the fact that security messages are very infrequent during normal execution and hence marginally affects the throughput and latency of the network.

5.2 Area/Power Evaluation

We implemented the hardware design using Verilog HDL and performed functional verification using ModelSim Verilog Simulator from Mentor Graphics. Synthesis was performed using Design Compiler from Synopsis along with the components from the Synopsis DesignWare IP library. We used 90nm technology library from TMS (TCBN90GHP). Power results are obtained from Design Compiler. The memory power data was obtained using CACTI power model [18].

Table 4: Security Core Metrics

Area [μm^2]	427320.57
Energy [μJ]	250.32

Hence the overhead in terms of area and power introduced by Security Core is minimal. However each of the SNI also contributes to the area and power as detailed in [7]. The addition of the Security Core provides enough confidence to detect and isolate any malicious application executing in SoC.

6. CONCLUSION

In this paper we design and implement a communication protocol following the biological immune system to address security in SoC. We also elaborate on the framework for a secure Network on Chip to support such security related communication on the chip. The security task is distributed among the SNIs and the security core supervises and configures all the security modules. In case of a compromised core being detected, the security core performs the task of isolating the core from the rest and migrate the tasks of the compromised core to other similar core if available.

7. REFERENCES

[1] L. Benini and G. De Micheli. Network on Chips: A New SOC Paradigm. IEEE Computer, 2002

[2] IBM, Global Business Security Index Report. <http://www.ibm.com> 2008

[3] J.Coburn, S.Ravi, A.Raghunathan, and S.Chakradhar. Seca: security-enhanced communication architecture. In CASES '05: Proceedings of the 2005 Int. Conference on Compilers, architectures and synthesis for embedded systems, pages 78–89, New York, NY, USA, 2005. ACM Press.

[4] C. H. Gebotys and Y. Zhang. Security wrappers and power analysis for SoC technology. In Proc. of CODES+ISSS'03, pages 162–167, Oct. 1-3 2003.

[5] J. P. Diguët, S. Evain, R. Vaslin, G. Gogniat, and E. Juin. NoC-centric security of reconfigurable soc. In Proc. Of NOCS'07, May 7-9 2007.

[6] L. Fiorin, G. Palermo, S. Lukovic, V. Catalano, and C. Silvano. Secure Memory Accesses on Networks-on-Chip. IEEE Trans. on Computers, 57(9):1216–1229, September 2008.

[7] L. Fiorin, G. Palermo, S. Lukovic, and C. Silvano. A Data Protection Unit for NoC-based Architectures. In Proc. Of CODES+ISSS'07, Sept. 30 - Oct. 5 2007.

[8] J. Niemela. F-Secure Virus Descriptions. F-Secure, December 2007.

[9] L. Negri. Power Modeling and Optimization for Wireless Networks. PhD thesis, Politecnico di Milano - Dipartimento di Elettronica e Informazione, 2006.

[10] Michael J. Lyons and David Brooks. The design of a bloom filter hardware accelerator for ultra low power systems. Proceedings of the 14th ACM/IEEE international symposium on Low power electronics and design 2009

[11] C. Ciordas, T. Basten, R. Radulescu, K. Goossens, and . Van Meerbergen. An Event-Based Monitoring Service for Networks on Chip. ACM Trans. on Design Automation of Electronic Systems, 10(4):702–723, Oct. 2005.

[12] D. Dasgupta, Ed., Artificial Immune Systems and Their Applications, Heidelberg, Germany: Springer-Verlag, 1999.

[13] G. B. Lamont, R. E. Marmelstein, and D. A. Van Veldhuizen, “A distributed architecture for a self-adaptive computer virus immune system,” in New Ideas in Optimization. New York: McGraw-Hill, 1999, Advanced Topics in Computer Science Series, ch. 11, pp. 167–183.

[14] D. Dasgupta, “An artificial immune system as a multi-agent decision support system,” in Proc. IEEE Int. Conf. Systems, Man and Cybernetics, Oct. 1998, pp. 3816–3820.

[15] E. Hart, P. Ross, and J. Nelson, “Producing robust schedules via an artificial immune system,” Proc. IEEE Int. Conf. Evolutionary Computing, pp. 464–469, May 1998

[16] K. Mori, M. Tsukiyama, and T. Fukuda, “Multi-optimization by immune algorithm with diversity and learning,” in Proc. Second Int. Conf. Multiagent Systems, Dec. 1996, pp. 118–123.

[17] <http://www.eecs.umich.edu/mibench/>

[18] CACTI, HP Labs, <http://www.hpl.hp.com/research/cacti>