SPP-NIDS - A Sea of Processors Platform for Network Intrusion Detection Systems

Luís Carlos Caruso, Guilherme Guindani, Hugo Schmitt, Ney Calazans, Fernando Moraes
Faculdade de Informática – PUCRS – Av. Ipiranga 6681, Porto Alegre, Brazil
{calazans, moraes}@inf.pucrs.br

Abstract

A widely used approach to avoid network intrusion is SNORT, an open source Network Intrusion Detection System (NIDS). This work describes SPP-NIDS, a architecture for intrusion detection supporting SNORT rules. SPP-NIDS is attractive to real-world network intrusion detection, due to its scalability, flexibility and performance features. A parameterizable cluster of simple processors provides system scalability. Hardware NIDSs described in the literature often employ hardwired comparators to verify if the incoming network traffic has data potentially containing intrusion attacks. Such NIDSs must be re-synthesized when a new set of rules is available, which happens frequently. In SPP-NIDS, the rule set defining network intrusion patterns is stored in RAM, enabling its straightforward upgrade. The proposed system, when implemented in a 2-million gate FPGA is able to work at a 100 Mbps network data rate, using the complete set of SNORT rules. If more performance is required, it suffices to scale the system, by adding extra processors.

1 Introduction

The threats to the goods of an enterprise by means of network attacks (also known as *network intrusion*) increase proportionally to the increase of the dependency of these enterprises on computer networks. The geometric growth of people and enterprises connected to the Internet and the facility of use of networks are not factors contributing to reduce this problem [1].

A widely used approach to avoid network intrusion is SNORT [2], an open source packet sniffer and packet logger network intrusion detection system (or NIDS), commonly located after a firewall, trying to detect complex network attacks [3]. SNORT works by comparing character patterns present in the network traffic to patterns defined by a set of rules, using pattern-matching algorithms. Employing off-the-shelf processors and/or general purpose computers based on standard Instruction Set Architectures for building NIDS cannot guarantee that all known rules can be timely verified, given the high throughput of current networks and the intrinsically sequential processing of these architectures.

The enhanced performance of electronic systems furthers research and development of specific system architectures to meet the performance requirements of computer networks. One domain where the need for such architectures is acute is network security systems.

Real world NIDSs present deficiencies, most of these related to sub-dimensioning and bad use of devices dedicated to prevent network intrusion. For example:

- NIDSs require high performance equipments and algorithms. They must be capable of searching and processing huge amounts of data at high throughput - sub-dimensioning here is not acceptable.
- NIDSs require frequent updates in their knowledge bases - some flexibility to recognize new types of attacks is also recommendable.

Given this picture, the importance of employing NIDSs, the need to use them correctly and their high performance exigency degree it is possible to depict the best choices to implement NIDSs as those presenting the following features:

- *Specialization* - the designer of a NIDS may restrict himself/herself to provide computing power only to the pattern detection related tasks;
- *Parallelism* - a NIDS design is expected to provide the maximum possible amount of parallel computing power, and possibly some way to guarantee support to the dynamic expansion of the system;
- *Adaptability* - a NIDS knowledge base must be easily expandable on the fly.

18th IEEE/IFIP International Workshop on Rapid System Prototyping(RSP'07)
0-7695-2834-1/07 $20.00 © 2007 IEEE

This work presents the SPP-NIDS, a system built with the above features defined as design requirements.

The rest of the paper is organized as follows. Section 2 presents related works in specialized NIDS. Section 3 presents the main features of the SPP-NIDS system, while Section 4 details the system modules. Sections 5 and 6 present prototyping results and system validation, respectively. Section 7 concludes this paper.

2 Related Work

Cho [4] implements 32-bit hardwired comparators in parallel. The synthesis frequency obtained for an Altera EP20K was 90 MHz, and multiplying it by 32, they argue their system is able to sustain 2.88 Gbps. The system was designed to process only 105 SNORT rules. This implementation can achieve high throughput for a small number of rules, however it is not scalable neither flexible. Sourdins [5] implements an architecture similar to Cho, using a register tree to improve performance. According to the authors, their architecture can be used in a 10 Gbps network. Backer [6] implements unary comparators. The key idea is the fact that the SNORT rules contains only 100 different bytes, enabling the implementation of simpler comparators. Carver [7] uses a different approach. Regular expressions are employed to reduce area and improve performance. They employ 8-bit comparators, and can achieve 1Gbps throughput.

The Snort version 2.0 includes the work of Coit [8], which improves the performance using the Boyer-Moore algorithm. Attig [9] uses hash functions generators, Bloom filters and a limitation of 26 characters per rule to detect matches.

Three common characteristics exist in the previous works: (*i*) limited number of rules, below the 2124 existing rules of the SNORT v 2.11; (*ii*) limited size of the rules; (*iii*) limited characters. The proposed architecture, SPP-NIDS, does not have such limitations. Even if the achieved throughput is inferior to the hardwired 32-bit comparators, SPP-NIDS considers not only the rule processing, but also the treatment of the attacks.

3 SPP-NIDS: Platform and Prototyping

Figure 1 depicts the block diagram of the SPP-NIDS and its external interfaces. SPP-NIDS is a platform with an architecture composed by an IDS controller processor (named MR2 [12]), an attached dedicated reconfigurable coprocessor, a host processor and interfaces to a link connecting external and internal networks.

The main part of the system is the dedicated coprocessor, named *GIOIA*, designed for performing SNORT rules pattern matching. *GIOIA* is a cluster of very simple custom identical processors, each of them named *picoCPU*. Configuring the amount of *picoCPUs* in *GIOIA* it is possible to satisfy specific IDS performance requirements, in a scalable way.

The cluster is reconfigurable because it allows downloading a distinct set of SNORT rules to each *picoCPU*. This can be changed at every detection cycle for all *picoCPUs*. Each *picoCPU* serially applies all rules contained in its configuration to every IP packet it receives.

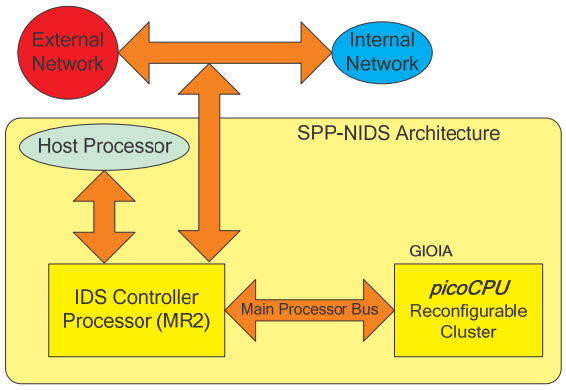

Figure 1. SPP-NIDS architecture block diagram.

Besides the cluster of *picoCPUs*, *GIOIA* contains: (*i*) one match register for each *picoCPU*; (*ii*) an interrupt generator; (*iii*) a bus responsible to carry the SNORT rules and the network data to the *picoCPUs*. After applying rules to data coming from the external network, a *picoCPU* either detects a suspicious pattern or not. In the first case, it writes data about the rule that triggered the matching in its match register. Otherwise, it remains idle until the next data coming from the network. The interrupt generator performs a daisy-chain search through all match registers. If any match occurs, it interrupts the IDS controller CPU.

The host processor is typically an external equipment (a PC or workstation) responsible for initializing the IDS Controller processor and the *GIOIA* cluster.

The first hardware prototype of this platform was targeted to the Digilent Spartan-3 System Board [10], with a 3-*picoCPU* cluster. Due to the input/output restrictions imposed by this prototyping platform, a series of adaptations of the SPP-NIDS architecture were needed.

First, the Spartan-3 System Board does not contain a built-in network interface, such as an Ethernet PHY module. Thus, the behavior of the network is emulated by using part of the MR2 data memory to contain IP packets that are read by specialized software routines running in the MR2 processor. This memory is implemented with FPGA internal Block RAMs. A next version of this prototype implementation will leave the task of generating network packets to the host processor, to increase flexibility.

Another feature of the implemented prototype is that the host processor is in fact a PC computer linked to the prototyping platform through a serial interface, using a RS-232 compatible protocol. This interface enables the host to reset and to hold the MR2 processor. In addition, it allows the host to load the MR2 instruction memory with the *GIOIA* cluster management software and load the MR2 data memory with SNORT rules to be applied by the *picoCPUs* and IP packets to emulate the network behavior. The host also uses this interface to receive information about potential invasion attempts in IP packets.

4 Detailed Description

Figure 2 presents the structure of the implemented prototype of the SPP-NIDS platform. The *GIOIA* cluster is mapped on the processor data memory. In this way, memory read and memory write MR2 instructions provide access to the cluster.

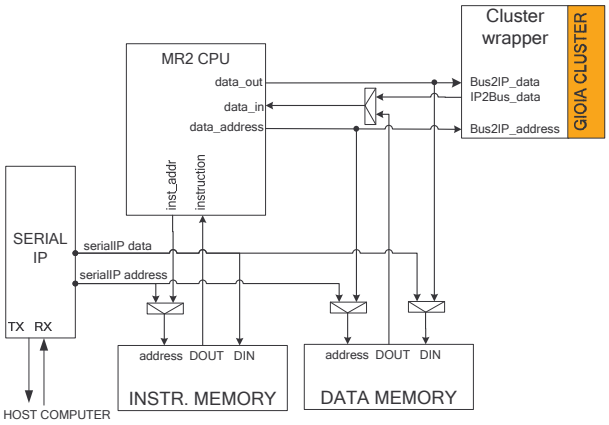

Figure 2 - Structure of the SPP-NIDS prototype.

4.1 The GIOIA Cluster

The set of rules in SNORT [11] is updated very often. Table 1 illustrates the number of rules on a per protocol basis for SNORT version 2.11. The larger set of rules belongs to the TCP protocol, and these rules are further subdivided in sub-classes, resulting in approximately six hundred rules for the largest sub-class. Thus, an IDS for this version of SNORT and supporting the most stringent demands of rule application must be capable of comparing each network input character against at least six hundred rules in the time available between the reception of two characters. This limit establishes a minimum design performance requirement on the *GIOIA* cluster.

Table 1 - Number of rules by protocol, SNORT v. 2.11.

Protocol	Number of rules
TCP	1784
UDP	165
IP	44
ICMP	131
TOTAL	2124

The cluster has 3 sets of signals, implementing 3 distinct interfaces: (*i*) a configuration interface, to load SNORT rules in the cluster; (*ii*) a packet interface, to receive network input data; (*iii*) an intrusion detection interface or simply intrusion interface, to inform the IDS controller CPU about rule matching. Figure 3 presents the external interface and the internal organization of the *GIOIA* cluster, abstracting the configuration interface.

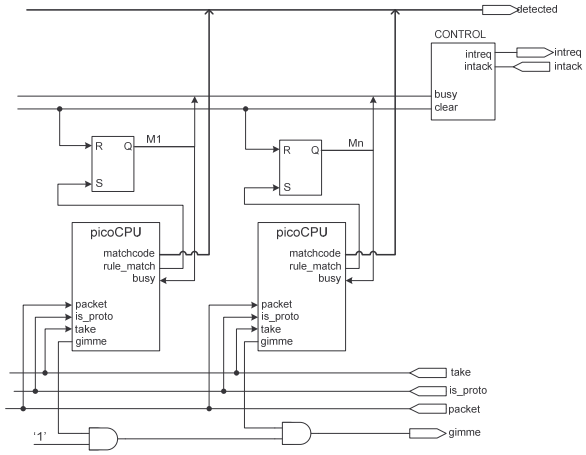

Figure 3 - Cluster internal organization and interface to the IDS controller CPU through the cluster wrapper.

The cluster packet interface can only receive a new character when all *picoCPUs* are available to process this new character. The wire *gimme* is used to signal this availability. This is why an important point in the cluster design is to balance the number of rules distributed to each *picoCPU*.

When a *picoCPU* detects an attack, it asserts signal *rule_match*, setting the match flip-flop (one of *M1* to *Mn*). Setting a given match flip-flop provokes the assertion of the *busy* signal of the corresponding *picoCPU*, holding it. The cluster control unit sweeps the match flip-flops data in a daisy chain, looking for a *picoCPU* that has detected an attack. When this occurs, an interrupt signal is sent to the MR2, the attack code (named *matchcode*) is stored in a cluster register (not shown Figure 3) and the *picoCPU* is released through signal *clear*. Note that after the interrupt request is asserted, the *picoCPU* that detected the attack is back to normal operation. The maximum time a *picoCPU* is hold from applying its attack detection algorithm in clock cycles is equal to the number of *picoCPUs* in the cluster minus one.

4.2 The GIOIA Cluster Wrapper

The cluster wrapper has the function of adapting the cluster signals to IDS controller CPU data bus, as illustrated in Figure 4. This bus allows access to the three cluster interfaces.

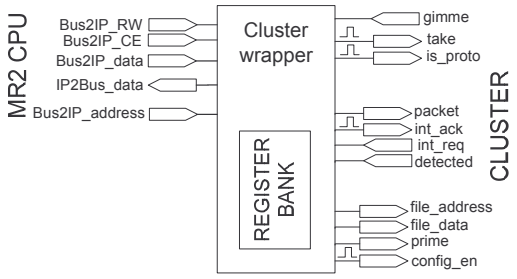

Figure 4 - Cluster wrapper structure.

The cluster configuration process, responsible to load the SNORT rules into the *picoCPUs*, takes place with the MR2 writing data to the wrapper internal register bank. After detecting a write operation on the configuration registers, the wrapper generates a pulse on signal *config_en*. The same procedure is followed for sending network input data to the cluster. A write operation in the registers allocated to perform packet data transmission to the cluster generates pulses in signals *take* (indicates packet data) and *is_proto* (indicates the protocol of a new packet). It is important to point that there is no need for the MR2 to initialize control registers, this action is undertaken by the wrapper.

The current implementation collects attack data using polling and not interrupts as described above. The MR2 software monitors the attack detection register and, once this is found to contain positive attack data, executes the attack treatment actions. Once the detection process is initiated, the MR2 asserts a pulse in the *int_ack* signal to release the cluster control. A future implementation of the system will substitute this by the interrupt mechanism described above.

4.3 The PicoCPU

The function of a *picoCPU* is to verify if certain patterns (SNORT rules) occur in some section of the input network traffic. *PicoCPUs* store in an internal RAM patterns against which the network traffic must be compared. The use of RAM to store patterns confers the property of reconfigurability to *picoCPUs*. Since SNORT is an evolving set of rules [11] this is an important feature of the SPP-NIDS platform. Every time SNORT releases new rules, these can be immediately accounted for in SPP-NIDS by just adding these new rules to the MR2 data memory.

The *picoCPU* implements in hardware a two-nested loop. The external loop is responsible for sampling one by one the characters of a packet. The internal loop compares each character against one of the bytes in every rule assigned to this *picoCPU*. One of four possible situations may occur during the comparison of a sampled character with a given byte of some rule:

1. no match, being the rule pointer at the start of the rule;
2. match, without reaching the end of the rule, in which case the rule pointer must be advanced;
3. no match, but the comparing rule must have its pointer reset to its starting position ;
4. match, being the pointer at the last character of the rule.

The standard situation is item one above, which is executed in a single clock cycle. Situations 2 and 3 take two clock cycles each. Situation 4 corresponds to rule matching. The treatment of this situation depends on the IDS controller CPU and on the cluster size. Since detections are bufferized, there is little performance penalty involved in case of attack detection, as already discussed at the end of Section 4.1.

Equation 1 defines the *picoCPU* throughput, in Mbps.

$$T_p = \frac{f * 8}{p_t * n_r} \quad (1)$$

where:

- *f*, the picoCPU frequency;
- *8*, the picoCPU word size;
- *pt*, represents the average time to process each character in a packet, in clock cycles;
- *nr*, the number of rules each picoCPU compares against each character.

As a case study, considers f=200 MHz, p_t=1.5, n_r=20. Data throughput (T_p) exceeds 53 Mbps. Assuming the most stringent rule set treatment requirement established in Section 4.1, 600 rules, a cluster with 30 *picoCPUs* operating in parallel is able to support the same performance computed for a single *picoCPU*, i.e., 53 Mbps.

4.4 Serial IP

The Serial IP Core is responsible to provide communication between the user working in a host computer and the intrusion detection system. This communication is performed by an RS-232 protocol standard serial interface.

The host computer is used to send: (*i*) the software to be used by the MR2 processor; (*ii*) the rules to the *picoCPUs*; (*iii*) the network flow - characters to be compared. The host computer receives from the system the number of attacks and the number of rules generating them.

4.5 MR2 CPU

The SPP-NIDS IDS controller CPU is called MR2. MR2 [12] is a 32-bit Harvard processor that partially implements the MIPS 2000 instruction set architecture. It is responsible for loading each *picoCPU* memory with patterns, as well as for controlling and monitoring *GIOIA*. It is also responsible for interacting with the host processor, and with the external and internal networks.

5 System Prototyping

The target device is the Spartan3 XC3S200. The SPP-NIDS system uses 62.58% of the available LUTs, 26.13% of the available flip-flops and 100% of the available BRAMs. Table 2 presents the area results after logic synthesis. The area of the three main macro modules (MR2 processor, serial IP and *GIOIA* cluster) is detailed. The Table also presents the area of each *picoCPU* (3 in this implementation). Note that this block is very small (150 LUTS, 102 flip-flops and 1 BRAM), enabling to implement large clusters, when more BRAMs are available.

Figure 5 displays the SPP-NIDS floorplanning after placement and routing. The placement of each macro block was constrained using the floorplanning tool, enabling to obtain an optimized routing.

Table 2 - Leonardo area report.

	LUTS	FFs	BRAM
picoCPU	150	102	1
Main macro modules			
MR2 processor	1169	274	9
Serial IP	323	243	0
GIOIA cluster	753	583	3
Other small blocks	158	39	0
TOTAL	2403	1139	12
Available in the device	3840	4359	12
Utilization	62.58%	26.13%	100.00%

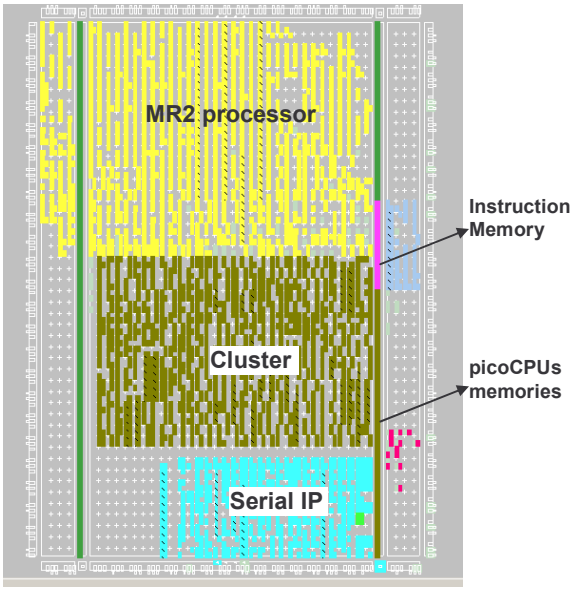

Figure 5 - SPP-NIDS floorplanning after placement and routing.

6 System Validation

To compile the SNORT rules the user employs the RuleWizard tool (Figure 6), developed in the context of this work. The tool compiles the ASCII description of the SNORT rules, generating: (*i*) set of rules to be stored in each *picoCPU*; (*ii*) random attack vectors.

The application developed for the MR2 processor is responsible to: (*i*) initialize the contents of the *picoCPUs* memories; (*ii*) send data packets to the cluster; (*iii*) monitor attacks. This application is merged to the data obtained with the RuleWizard tool, resulting in the MR2 objected code.

Before executing the system in the FPGA, it can be simulated using the ModelSim tool. Figure 8 displays how to identify the individual alerts for each *picoCPU*, through signal *match*. Observe in Figure 8 that each

picoCPU detects four matches: rules {1,7,9,6} in *picoCPU1*; rules {1,6,A,8} in *picoCPU2*; rules {1,12,B,10} in *picoCPU3*.

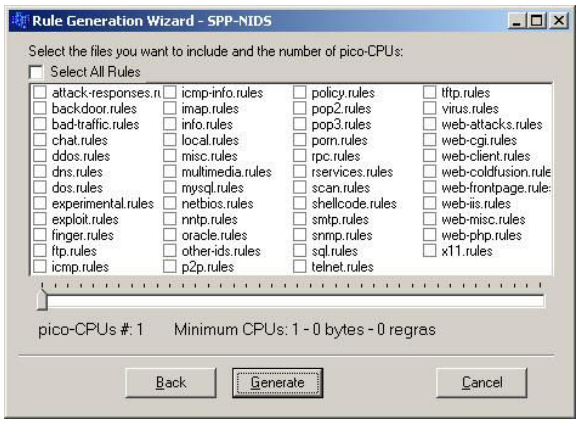

Figure 6 - *RuleWizard* tool.

In the upper part of the Figure 8 we can observe the signal *intreq*/*detected* signaling an alert to the MR2 CPU. The signal *intack* releases the cluster control. Note also the *busy* signal at each *picoCPU*. Each *picoCPU* waits in average two clock cycles after a rule matching.

The hardware validation starts by downloading the SPP-NIDS bitstream into the prototyping board. A second tool developed in the context of this work enables the user to start the SPP-NIDS system and collect the resulting attacks from the prototyping board. Figure 7 illustrates the trigged rules. Compare this result to the waveforms of Figure 8, where each *picoCPU* has detected four rules. The set of rules detect in simulation are the same of the ones obtained in the hardware execution, showing the correct system operation.

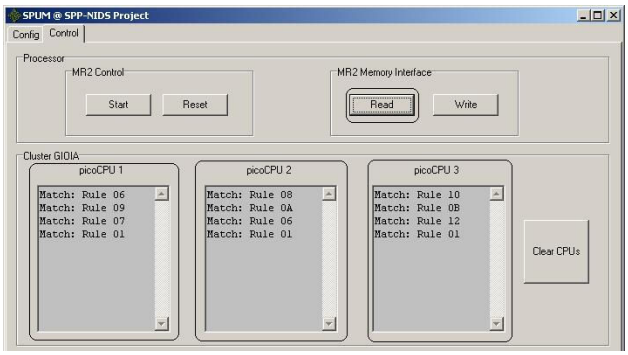

Figure 7 - Triggered Rules.

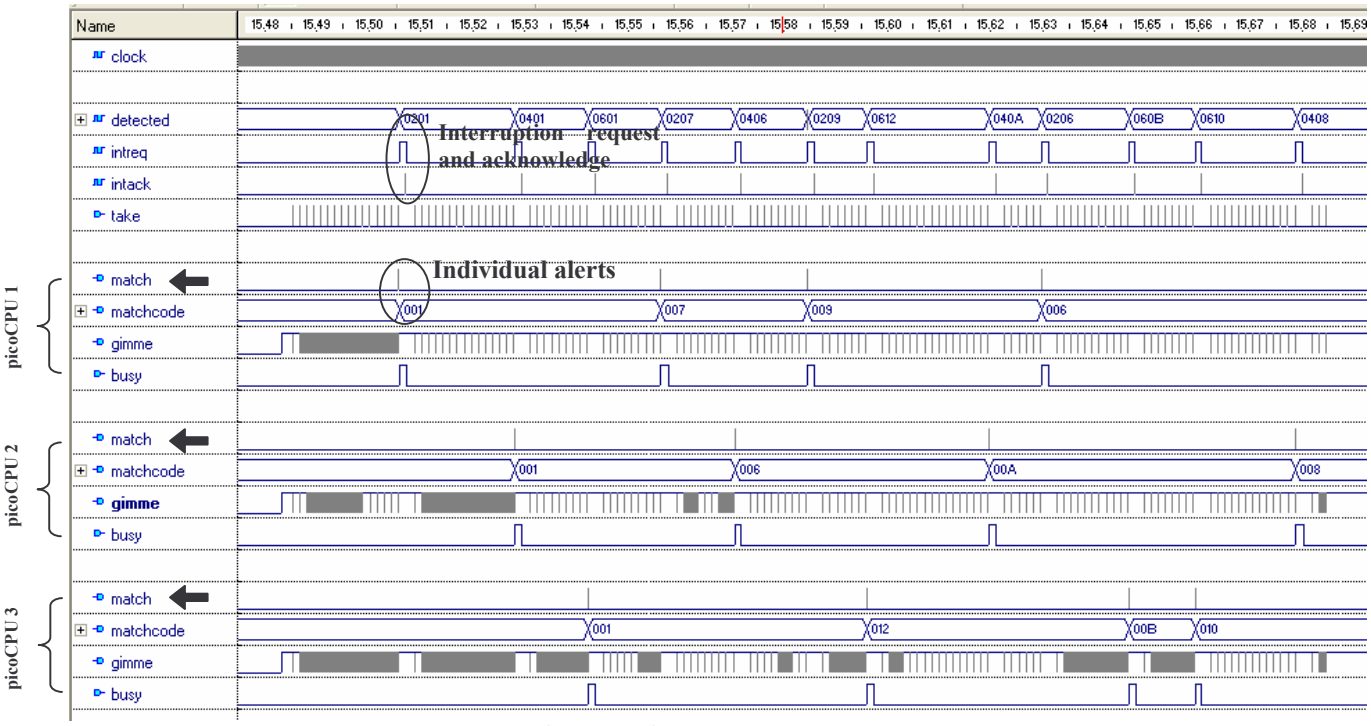

Figure 8 - SPP-NIDS Functional validation.

7 Conclusions and Future Works

This work presented a proof-of-concept NIDS based on the SNORT rules. This system, named SPP-NIDS, combines flexibility and performance. Flexibility comes from the use of embedded memories to store the rules, instead to hardwired comparators, as found in the literature [4][5]. Performance comes from the spatial parallelism of the *picoCPUs*. Using a two-million gate FPGA it is possible to construct a cluster with 40 *picoCPUs*. Such cluster could easily work at 100 Mbps, verifying every packet in a 100 Mbps Ethernet flow.

Future work in the SPP-NIDS includes: (*i*) connect the alert interface to the CPU interrupt signal; (*ii*) connect the packet interface to a realistic network interface, e.g. Ethernet; (*iii*) migrate to a system with two embedded CPUs, one for network input data preparation for the *GIOIA* cluster and the second one for intrusion detection treatment; (*iv*) work with an heterogeneous cluster, with specialized *picoCPUs*.

It is also possible to convert this proof-of-concept prototype in a real product. For example, considers the two PowerPC Virtex-II Pro XCVP30 device. A MAC IP and one PPC can be used to obtain network input data, execute the basic SNORT procedures, and send the packets to the *GIOIA* cluster. The second PPC may process the cluster alerts and send them, for example, to a firewall.

8 References

[1] Allen, J.; et al. "*State of the Practice of Intrusion Detection Technologies*", 2000. http://www.cert.org/archive/pdf/ 99tr028.pdf.

[2] SNORT. "*About SNORT Homepage*". Available at http://www.snort.org/ about_snort/, July, 2005.

[3] Beale, J.; et al. "*Snort 2.0 Intrusion Detection*". Syngress Publishing, Inc, Rockland, MA. 2003. 559 pages.

[4] Cho,Y.H.; et al. "*Specialized Hardware for Deep Network Packet Filtering*". In: FPL 2002.

[5] Sourdis,I.; Pnevmatikatos,D. "*Fast, Large-Scale String Match for 10Gbps FPGA-based Network Intrusion Detection System*". In: FPL 2003.

[6] Backer,Z; Prassana,V. "*Automatic Synthesis of Efficient Intrusion Detection Systems on FPGAs*". In: FPL 2004.

[7] Carver, D; et al. "*Assisting Network Intrusion Detection with Reconfigurable Hardware*". In: FCCM02, 2002, pp.111-120.

[8] Coit, C.; et al. "*Towards Faster String Matching for Intrusion Detection*". In: DARPA Information Survivability Conference & Exposition II, 2001, pp.367-373.

[9] Attig,M.; et al. "*Implementation Results of Bloom Filters for String Matching*". In: FCCM 2004.

[10] Xilinx, Inc. "*Spartan-3 Starter Kit Board*". User Guide, Mar., 2005.

[11] SNORT. "*SNORT Rules Homepage*". Available at http://www.snort.org/rules/, July, 2005.

[12] Calazans,N.; et al. "*MR2 Multicycle Processor*". Nov, 2004. http://www.inf.pucrs.br/~calazans/undergrad/ orgcomp/arq_MR2.pdf.