Evaluation of SNMP-like protocol to manage a NoC emulation platform

Otávio Alcântara de Lima Jr, Virginie Fresse, Frédéric Rousseau

Evaluation of SNMP-like protocol to manage a NoC emulation platform

Otávio Alcântara de Lima Jr. and Virginie Fresse
Hubert Curien Laboratory UMR CNRS 5516
Jean Monnet University - University of Lyon, PRES Lyon
Saint-Étienne, France

Frédéric Rousseau
TIMA Laboratory, UJF/CNRS/Grenoble INP, SLS Group
Institute Polytechnique de Grenoble
Grenoble, France

***Abstract*—The Networks-on-Chip (NoCs) are currently the most appropriate communication structure for many-core embedded systems. An FPGA-based emulation platform can drastically reduce the time needed to evaluate a NoC, even if it is composed by tens or hundreds of distributed components. These components should be timely managed in order to execute an evaluation traffic scenario. There is a lack of standard protocols to drive FPGA-based NoC emulators. Such protocols could ease the integration of emulation components developed by different designers.**

In this paper, we evaluate a light version of SNMP (Simple Network Management Protocol) to manage an FPGA-based NoC emulation platform. The SNMP protocol and its related components are adapted to a hardware implementation. This facilitates the configuration of the emulation nodes without FPGA-resynthesis, as well as the extraction of emulation results. Some experiments highlight that this protocol is quite simple to implement and very efficient for a light resources overhead.

I. Introduction

The accelerated development of the VLSI microelectronics technology enables the integration of ten or hundred of complex logic cores in a System-on-Chip (SoC). Those SoCs consist of a set of processors, memories, mixed signals components and specialized logic blocks that share a communication infrastructure. The growing communication requirements of those systems demand scalable, flexible and parallel communication architectures. The Networks-on-Chip (NoCs) emerge as the most promising communication technology for the modern many-cores SoC, whereby they have greater scalability than other solutions such as buses and point to point connections.

The NoCs have a large design space concerning different aspects like topology, routing, flow control, QoS, among others. A SoC designer must make several architectural choices and performance evaluations to find the best NoC configuration for a given project. Furthermore, all those system aspect must be tested and validated. An important part of a SoC design costs and time is spent in design space exploration and validation of the communication architecture. NoC specialized design tools can be used to reduce the design costs.

High level simulation [2], [4], [1] can be used in the early stages of development process for a faster, however less accurate, design space exploration. Usually those simulations are written in high level languages like C, C++ and Java. Cycle Accurate and Byte Accurate (CABA) simulations [12] have higher accuracy in detriment of very high execution times. They are used to validate an HDL model.

The FPGA-based emulation platforms [3], [9] are a promising technique for NoC design space exploration and benchmarking. Those platforms can drastically reduce the design time and they also possess high accuracy. Although the hardware NoC emulation have some disadvantages. The platform synthesis time is an obstacle, because any change in the target NoC demands the platform re-synthesis. Some approaches [4], [15] use the partial reconfiguration capabilities of state of art FPGAs to bypass this situation. Another drawback is the FPGA resources limitations which determine the maximum NoC size that can be emulated. The multi-FPGAs platforms [5], [17] can be used to extend the amount of available resources. However the off-chip communications can represent a bottleneck.

The previous proposals of FPGA-based emulation platform do not offer emulation components interoperability. There is a lack of standard protocols targeted at providing common interface to manage the components of NoC emulators. The adoption of a standard protocol can ease the integration of emulation components created by different designers. Moreover, this protocol shall provide a common interface to the emulation components from the perspective of the management software running at the host PC in charge of the emulation.

In this paper, we evaluate the SNMP (Simple Network Management Protocol) to manage an FPGA-based NoC emulation platform. The evaluated communication model enables managing an emulation platform's components using a set of standard operations executed in associated Management Information Base (MIB). The MIB is a bank of registers holding the configuration data of a traffic scenario and its results. The operations on the MIB allow changing the platform behavior, in such a way that it is possible to configure and execute several evaluation traffic scenarios without FPGA resynthesis. Although the SNMP was conceived for computer networks, it can be adapted to the VLSI environment. In [7] the SNMP is used to manage a testing platform for SoCs, which use the P1500 test standard.

The SNMP has been chosen for managing a NoC emulation platform because of its simplicity and its flexibility to deal with devices from different manufacturers. Indeed, to manage a new device using SNMP it is needed only to known the device's MIB structure. Moreover, the SNMP RFCs has standardized how a MIB description should be written, which eases the management of SNMP enabled devices as well as the adoption of components from different designers. In this paper, the SNMP protocol and its related components are adapted to a hardware implementation, which is is quite simple to implement and

very efficient for a light resources overhead as highlighted by some experiments.

The rest of this paper is organized as follows. The section II presents the related works. The section III introduces the concepts of network management based on SNMP. Afterward, the emulation platform is presented in section IV. The evaluation experiments are presented in section V. At last, the section VI presents the paper conclusions.

II. Related Works

The FPGA-based emulation platforms can be used to accelerate NoC benchmarking as well as design space exploration. Those platforms have high accuracy and low execution time in relation to NoC simulators. The NoC hardware emulation has some drawback. For instance, the platform synthesis time represents a major problem, because any change in the target NoC demands the FPGA re-synthesis. However, the FPGA partial reconfiguration can be used to bypass these limitations. Furthermore, a platform shall also provide ways to execute different traffic scenarios without FPGA re-synthesis.

An FPGA-based emulator uses specific components to inject and to extract packets in and from the network. The Traffic Generators (TGs) are responsible for generating traffic based on applications or synthetic models. The Traffic Receptors (TRs) are responsible for retrieving the packets and computing performance metrics. Those components can be implemented in hardware or software. Some proposals [9], [15] execute software implementations of the TGs and TRs in a dedicated processor. This approach enables easy configuration of the emulation platform. However the communication between the dedicated processor and the NoC is considered as a bottleneck. A single processor cannot emulate the traffic generated by all application cores for large NoCs at high offered loads, neither it can emulate the parallel nature of the application cores.

In opposite, the hardware implementation of traffic related components occupies valuable FPGA resources, but it can ensure higher communication bandwidth, because the emulation components can be directly connected to the NoC. However it is harder to configure the emulation platform because its components are physically distributed. The Figure 1 represents this organization for a simple 2x2 mesh NoC. In this architecture a host PC is responsible for configuring the TGs and for retrieving the emulation results from the TRs. In this example the host PC configures the TG connected to the router 11 (red arc) and it retrieves the traffic results from the TR connected to the router 01 (blue arc). The proposed SNMP-based platform uses a similar approach, which is explained in section IV.

In the rest of this section, some propositions of FPGA-based NoC emulators are presented. The AcENoCs [9] is a flexible and cycle-accurate FPGA emulation platform for validating synchronous and GALS-based NoC architectures. The platform is divided into two frameworks. The hardware framework comprises the interconnection network to be emulated. The software framework is implemented in a soft core processor and it consists of logical implementation of traffic generators, source queues and traffic receptors. The connection between the two frameworks can represent a performance bottleneck.

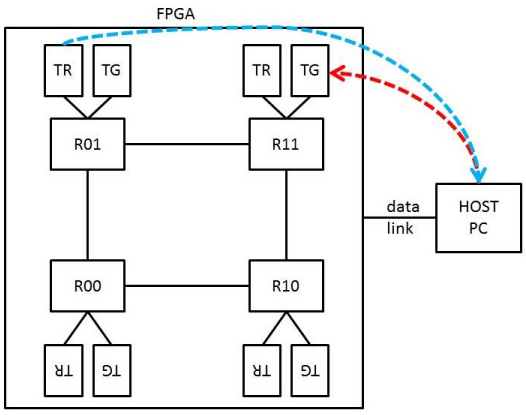


Fig. 1: hardware implementation of TGs and TRs.

The DART [14] is an FPGA-based NoC emulation platform that enables different NoC architectures to be simulated without recompiling/resynthesizing the DART engine. The routers are organized in partitions and the partitions are connected through a crossbar switch. DART simulates a NoC by mapping the simulated NoC to DART partitions and by reconfiguring the crossbar. The DRNoC [6] is a NoC emulation platform. This platform is based on FPGA partial reconfiguration capabilities. Therefore, it enables the dynamic insertion or removal of routers and cores in a mesh NoC.

The previous proposals of FPGA-based emulation platform do not address the issues related to the interoperability of emulation platforms components. The adoption of a standard protocol can ease the integration of emulation components created by different designers. In order to execute an emulation scenario, all the emulation components must be configured and, afterward, they must be accessed to extract the emulation results. The proposals that use hardware TGs and TRs usually apply a set of independent buses to connect them [3]. In opposite, the software implemented TGs and TRs can use software libraries to share data, but they must share a single bus to access the network routers [9], [15]. In both cases, there is no standard interface to manage the emulation components from the host PC perspective.

The management of the TGs and TRs imposes challenges. It must allow dynamic configuration and easy extraction of the emulation results. In this paper, we evaluate the SNMP communication model to manage an FPGA-based NoC emulation platform. The adoption of already tested management concepts enables taming the complexity of the NoC emulation platform. Moreover, the SNMP creates a common interface to manage emulation components from different manufacturers from a host PC.

III. SNMP

As connected computer systems become critical for modern business environment, monitoring and ensuring their reliability in performance is absolutely required. That is the reason why the IETF developed the SNMP protocol in the ends of the 80s. Until today, this protocol is the reference for network management because it is simpler than others solutions.

The SNMP is an application level protocol of the TCP/IP stack. It is a standard Internet protocol and its first version is defined in the RFC 1155, RFC 1212 and RFC 1157. The first two of these RFCs describe the SNMP data definition language, which specifies how the MIB must be described. The third one describes the protocol operations. The other two versions of this protocol are also defined by RFCs. In the context of this paper, we are concerned only with the aspects of the first version of this protocol.

The SNMP architectural model is a collection of network management stations (managers) and networks elements (managed devices). The network elements are devices connected on the network like routers, gateways, among others. Each managed device contains software called agent, which is responsible for handling the manager's requests. The SNMP is a protocol to monitor devices. In addition, it is capable of dealing with the configuration tasks and to change settings from a distance.

The MIB defines the structure of the management data of a device. It uses a hierarchical namespace containing the object identifiers (OID). Each OID can be read or written by the operations provided by the SNMP. The MIB can be described as a repository which all managed objects of a device are listed in a standardized tree hierarchy. The Figure 2 depicts the SNMP architecture.

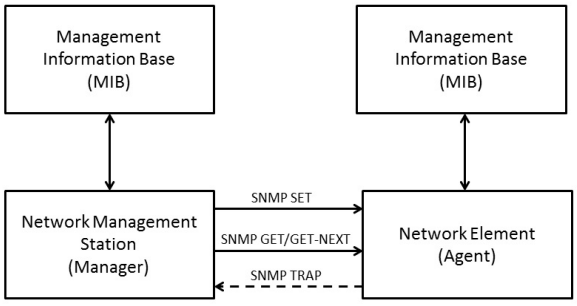


Fig. 2: SNMP architecture and operations.

The communication model applied between a manager and a device is the request-response model. A manager can access the objects stored in an agent's MIB through the SNMP GET and SNMP GET-NEXT operations. An agent replies those SNMP operations by sending a SNMP GET RESPONSE filled with the value of the required object. The SNMP SET enables to change the value of a MIB object. The SNMP was designed as a monitoring protocol. In order to give reactiveness to the protocol, the SNMP designers added the SNMP TRAP operation. This operation enables an agent to notify a manager about a specific device event. The SNMP TRAP is sent without previous manager's request.

In this paper, we evaluate the SNMP architecture to manage an FPGA-based NoC emulation platform. A communication protocol between a host PC and a target FPGA is designed. This protocol is inspired by the SNMP, but it is not aimed at a TCP/IP implementation. However, it is capable of configuring the emulation components and, then, extracting the emulation results. The emulation components are organized as SNMP agents. They are connected to a communication bus independent of the target NoC. The emulation components' MIB has all the registers needed to control their behavior during the execution of the traffic scenarios. Others management protocols such as SMBus [18] or NC-SI [19] could be solutions for integrating TGs and TRs. However, these protocols are designed specifically for computer motherboards management. They lack flexibility to manage a wide range of devices unlike SNMP that is a general purpose network management protocol.

IV. EMULATION PLATFORM

A. Architecture Overview

The top level architecture of the proposed emulation platform is depicted in Figure 3. The emulation platform is composed by synthesizable hardware components and by software components. The hardware components are meant to be implemented in a target FPGA, while the software components are meant to be executed in a host PC.

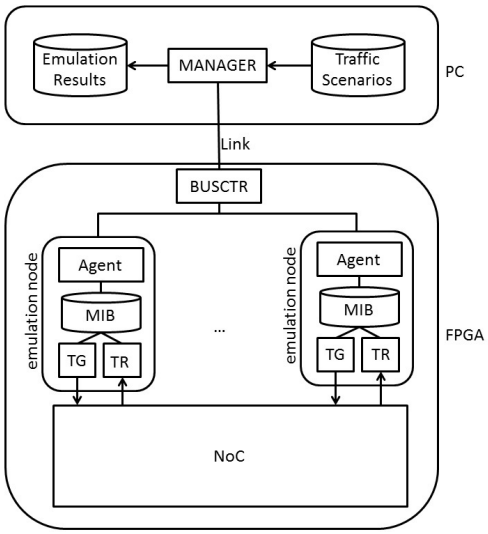


Fig. 3: architecture overview.

At first, we present the hardware components. The emulation node is a hardware implementation of a manageable device and it is directly connected to a NoC router. It is composed of four components: agent, MIB, TG and TR. The agent is responsible for decoding and executing the management commands. The MIB is an addressable register bank, which contains all the control and status registers that control the behavior of the traffic scenario generation, as well as they store the emulation results. The address of each MIB register is equivalent to the OID onto the standard SNMP implementation. In order to reduce the MIB resources requirements, it is necessary to limit the number of destinations of each node and consequently the number of registers spent on holding the emulation results. The goal is to keep the MIB size independent of the network size. The MIB is accessed by the traffic components through a simple interface, which enables reading and writing on the MIB's registers.

The two last components are the TG and TR that are responsible for generating and retrieving data on the traffic

scenarios. Those components were adapted from the emulation platform presented in [13]. The TG is connected to the router local input port. The TR is connected to the router local output port. In this implementation, the traffic is generated according to micro-benchmark models like transpose traffic, bit reverse, among others. Furthermore, it is possible to create traffic scenarios with an arbitrary number of source and destination nodes, including scenarios described as tasks graphs. The TR processes the traffic and it produces synthetic information about the performance of each communication link. The subsection IV-C presents the traffic models offered by this platform.

The adoption of the SNMP can ease the integration of TGs and TRs created by different designers. Indeed, to integrate a new component on this architecture it is needed only the description of the component's MIB. Moreover, the component's MIB can be described using the standards already defined in the SNMP RFCs.

The emulation nodes are connected on a dedicated communication bus, which is controlled by the BUSCTR component. This component is responsible for interfacing the platform manager and the managed devices. The BUSCTR decodes the management packets sent by the manager and it retransmits those to the target emulation node through a dedicated communication bus. This internal bus can be implemented using any standard on-chip buses protocols. A multi-hierarchical bus could also be applied according to the number of emulation nodes.

The software components are the manager, traffic scenarios and emulation results. The manager is implemented as a C library that communicates with the emulation platform over a serial link. This library enables easy integration of the emulation platform in third party systems. The communication protocol applied is inspired on the SNMP, but it is not targeted for a TCP/IP implementation. The traffic scenarios component is a set of test cases that may be generated by an external tool. The last component is the emulation results component, which is a synthesis of the performance results gathered during the emulation process. Those results can be used to the design space exploration, benchmarking or validation of a NoC implementation.

B. Protocol implementation

The SNMP has a proven management model based on a set of simple operations exchanged in a request-response model. This model is adapted to create a hardware implementation to manage a NoC emulation platform. In this model, the emulation nodes are represented as distributed components which can be managed through their MIBs. The SNMP provides the operations to handle the MIB from the manager side.

In order to build this platform, three SNMP standard operations are used and three new operations are added. The GET, GET RESPONSE and SET operations are standard SNMP operations that are adapted to this emulation platform context. The GET enables the manager to request the value of a specific object in an emulation node's MIB. This object can represent a traffic scenario parameter. The GET RESPONSE is the response generated by an emulation node to a GET packet. The SET enables the manager to alter the value of a specific object allocated in a MIB. In order to ease the control of the emulation platform, three new operations are added. The GO starts the emulation process. The RESET puts the emulation platform in the initial state. The EMU_END indicates that the manager can retrieve the emulation results.

The Figure 4 depicts the protocol packet format. All packet fields are 8-bit long. The first field represents the start of a packet. The OPER field indicates the requested management operation: GET, GET RESPONSE, SET, GO, RESET and EMU_END. Afterward, the emulation node address and the object identifier are represented in the next three fields. The OID indicates the address of a MIB register, which controls a specific behavior of the emulation node. The PARAM field is the parameter of the SET operations. At last, the CHECKSUM is an error control field. An incorrect packet is discarded and the destination requires its retransmission.

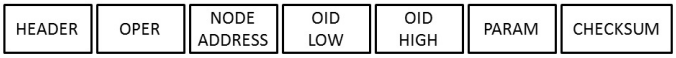


Fig. 4: packet format

C. Traffic Model

In order to evaluate a NoC performance, an emulation platform must be capable of generating rich traffic scenarios. The SNMP model can be easily adapted to many traffic models, but in this work it is used a micro-benchmark traffic model and a task graph traffic model. The proposed traffic models are implemented as two distinct hardware modules, described by two distinct MIBs. The platform user must select which traffic model will be used before the platform synthesis. The traffic model choice is based on the goals of the NoC evaluation.

The first traffic model enables the creation of micro-benchmark synthetic traffic as transpose, bit-complement, bit-reversal, among others. These synthetic traffic scenarios are largely used to evaluate the NoC, however they do not resemble to real applications. The second traffic model enables creating richer traffic scenarios based on task graphs, which can be used to model application behavior. Furthermore, a resource analysis of both implementations is presented in the subsection V-C.

D. Execution Flow

In this subsection, we explain the execution flow associated with the proposed NoC emulation platform. The execution flow is depicted in the Figure 5. In order to evaluate the performance of a NoC, a designer must specify a set of traffic scenarios. In the context of this paper, a traffic scenario describes how each emulation node must generate and receive packets. In this way, for each emulation node the traffic scenario specifies the amount of packets transmitted to each destination node, as well as the packets size, the associated injected charge, among others parameters. That stage is called the conception step. After this step, the execution flow enters in a loop to emulate sequentially all the traffic scenarios. Only one traffic scenario is kept in the MIB any time.

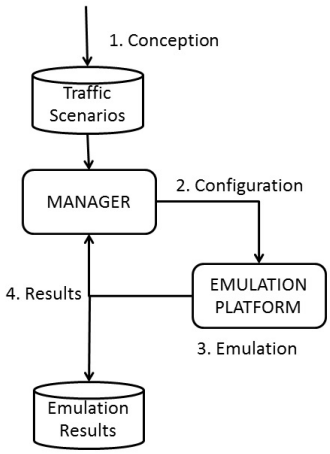


Fig. 5: execution flow.

The manager library can be used to transform a traffic scenario in the specific operations needed to configure the emulation platform. At first, each emulation node must be configured with the information gathered from the traffic scenario. This stage is known as the configuration step. On this step, the manager sends SET packets to change the value of objects stored in the MIB of each emulation node. The updated MIB changes the behavior of the associated node according to the current traffic scenario.

In order to accelerate the configuration step, the manager keeps a copy of each router's MIB and it only updates the MIB registers that have distinct values from the values specified by the previous traffic scenario. For instance, after executing the first traffic scenario, the manager starts the configuration step of the second traffic scenario. Then, the manager sends SET packets to change only the MIB registers that are distinct in both traffic scenarios. Those MIB registers that possess the same value are not changed. Thus, the number of SET packets sent is reduced. This feature is called differential MIB update and it is experimentally evaluated in the section V-B.

After the configuration step, the manager can send a GO packet and the emulation nodes start sending and receiving data packets. It is the emulation step which lasts as long as necessary to finish all the communication flows described by the traffic scenario. After the emulation nodes receive all the data packets, the emulation platform sends an EMU_END to the manager and the results step begins. On this step, the manager uses GET packets to retrieve the performance results. After this, the results are stored in the emulation results component. The execution flow must continue until all the traffic scenarios are emulated. Then, the emulation results can be analyzed in order to find out the NoC performance.

E. Implementation Details

This subsection presents the implementation details of the proposed emulation platform. At first, the SNMP hardware components are detailed and, afterward, the SNMP manager library is described.

1) SNMP Hardware Components: In this subsection, we describe the hardware implementation of the SNMP components: BUSCTR, Agent and MIB. These components are implemented in VHDL using RTL behavioral description.

The BUSCTR treats the SNMP packets from the manager and it adapts them to the internal bus format. This component has an asynchronous full-duplex serial interface to communicate with the manager and it also has a parallel bus interface which is capable to transfer 2 B per bus cycle. In order to execute one SNMP operation, the BUSCTR selects the target emulation node and then it takes five bus cycles to transfer the operation and its parameters to the target emulation node. The SNMP operations are executed sequentially by this component. A state machine implements the BUSCTR actions.

The Agent is responsible for decoding and executing the SNMP operations. It has a slave interface to the internal parallel bus and a custom interface to a MIB component. This interface enables the Agent to read and write in the MIB. A state machine is responsible for implementing the Agent behavior.

The MIB is implemented as a pair of registers banks. The first bank enables read operations and the second one enables write operations. These banks have distinct address spaces. The MIB is connected to a TG and a TR component through a simple interface, which enables the write and read of the MIB's registers.

The evaluation of the timing performance for executing each SNMP operation is presented in section V-A.

2) SNMP manager library: In order to drive the emulation platform, a SNMP manager library written in C is available. This library can be easily incorporated into existing and new NoC evaluation tools as a mean to drive the emulation. This subsection outlines the library functions as shown in Table I.

To start the communication with the emulation platform, the *emuNoC_initCOM* function must be called. Afterward, the platform must be reseted before starting an emulation scenario. The *emuNoC_Reset* lets the platform ready for use and it must be called before each emulation scenario. The user can create a scenario based on micro-benchmark with the *emuNoC_scenario* function or a custom scenario can be created using the *emuNoC_confNode* function, which must be called for each node participating in this traffic scenario. After the configuration of the emulation nodes, the *emuNoC_GO* function can be called to start the emulation. This function returns after the end of the emulation. Then, the performance results can be extracted using the *emuNoC_getLatency* function. Afterward, a new traffic scenario can be executed or the communication with the platform can be closed using the *emuNoC_closeCOM* function.

V. EXPERIMENTS

In this section, we describe experiments conducted in order to evaluate our proposal. The proposed NoC emulation platform is implemented in a ML605 Virtex 6 evaluation board for resources analysis and for performance evaluation of the SNMP operations and of the MIB update. The Xilinx ISE 14.3 and the XST are used to synthesize the emulation platform. The target NoC used in these experiments is a Hermes NoC

TABLE I: SNMP manager library

Operation	Description
emuNoC_initCOM	Initializes the communication with the emulation platform
emuNoC_Reset	Resets the emulation platform
emuNoC_confNode	Configures the communication requirements for one emulation node
emuNoC_scenario	Configure one microbenchmak scenario for this platform
emuNoC_GO	Initializes the NoC's emulation
emuNoC_getLatency	Retrieves the average latency for a pair of source and destination nodes
emuNoC_closeCOM	Closes the communication with the emulation platform

[10], in which the XY routing and handshake flow control are applied, as well as a 16-flit buffer is used by each router input port.

A. SNMP Operations Evaluation

The evaluation of the timing performance of the SNMP hardware components is described in this subsection. In order to evaluate this protocol, a workload based on three applications described as task graphs and three pseudo-random acyclic task graphs are executed on a 7x7 NoC. The applications used are MPEG-4, VOPD and multispectral imaging. The synthetic task graphs are created using the TGFF tool [16]. Each task graph is executed ten times in order to evaluate the emulation platform. The traffic scenario execution time is measured as well as the number of executed GETs and SETs, which enables calculating the time spent on the execution of the SNMP operations. From experiments, it is known that each GET takes 29 clock cycles, each SET takes 30 clock cycles and the circuit frequency is 66 MHz.

The SNMP related measurements are relative only to the execution of those operations by the SNMP dedicated circuits. It does not include the SNMP packet transmission delay, which depends on the link bandwidth between the manager and the FPGA board. Indeed, it does include the SNMP packet handling time, the internal bus latency and the emulation node's execution time.

The Table II depicts the amount of time needed to execute the SNMP operations as well as the average time needed to execute each traffic scenario within a 95% confidence interval displayed in parenthesis. Furthermore, the number of executed GETs, SETs operations and the number of tasks for each application are also presented.

The SNMP operations take only a very small fraction of the time spent to carry out a traffic scenario. For instance, for the MPEG-4 application the SNMP operations represent only 0.03% of the traffic scenario time and only 0.004% for the VOPD. Indeed, the SNMP operations are executed by lightweight circuits and they represent a very low overhead to drive the emulation tasks.

The time spent to execute the traffic scenario is much larger for the task graphs based on real applications. In this case, these applications have more bandwidth requirements than the synthetic ones. However, the time spent on the SNMP operations does not depend on the traffic scenario execution time. Moreover, there is a correlation between the number of tasks and the time spent on SNMP operations. More tasks lead to more time spent on the platform configuration.

The number of executed SETs operations is 36% - 47% bigger than the number of executed GETs for the three task graphs based on real application. For the synthetic task graphs, this difference is larger, it is around 55% - 58%. It is expected that the number of executed SETs is bigger than the number of executed GETs for a regular traffic scenario.

TABLE II: SNMP operations execution time

Application	SNMP [ms]	Traffic [ms]	No. of GETs	No. of SETs	No. of tasks
MPEG-4	0.15	430.1 (0.35)	104	232	12
VOPD	0.08	1768.1 (0.19)	52	142	12
Multispectral	0.11	895.2 (0.26)	80	169	14
TGFF 0	0.04	10.7 (1.57)	36	65	9
TGFF 1	0.11	6.3 (0.29)	92	156	14
TGFF 2	0.06	7.0 (0.41)	52	189	7

B. MIB Update Evaluation

In order to evaluate the performance of the differential MIB update which is described in section IV-D, we analyze the amount of data exchanged between the manager and the target FPGA to carry out the evaluation of a NoC. In this experiment, we execute a set of ten transpose traffic scenarios following the execution flow presented on IV-D.

Those scenarios are distinct only because of the injected charge which is consecutively incremented of 10% for each traffic scenario. The first traffic scenario has 10% of injected charge and the last one have 100% of injected charge. We measure the amount of data transferred to carry out these scenarios for different sizes of NoC. We conduct these experiments with the differential MIB update enabled and also when this features is disabled (linear MIB update).

The Table III presents the results. The differential MIB update transfers about 54% less data bytes than the linear MIB update. For a 115200 bps serial link and a 4x4 NoC, it takes about 405 ms to execute all the data transfers needed for the ten traffic scenarios when the differential MIB update is applied. In opposite, when the linear update is applied, it takes about 755 ms to accomplish this task. For each router added to the emulation platform, there is an average increment of 33% in the amount of bytes transferred for both update methods.

TABLE III: MIB update evaluation.

NoC size	Differential update (bytes)	Linear update (bytes)
4x4	5845	10885
3x3	3479	6314
2x2	1589	2849

C. Resources Analysis

As described in the subsection IV-C, the proposed emulation platform offers two distinct implementations of traffic models. The first one enables the creation of micro-benchmark synthetic traffic scenarios. The second one enables the creation of traffic scenarios based on tasks graphs. Both implementations share the same SNMP architecture, but they use different TGs and TRs components.

At first, we analyze the resources occupation of the micro-benchmark synthetic traffic implementation. The Figure 6 depicts the FPGA occupation for eight different configurations of

the NoC and the emulation platform. The resources occupation of the emulation platform comprises the implementation of the SNMP components, the traffic model components as well as the NoC itself. We observe that for a given NoC size the emulation platform has 2.8 times more LUTs and 4.7 - 5.3 times more Registers than the implementation of only the NoC components. Furthermore, these implementations do not use RAM resources.

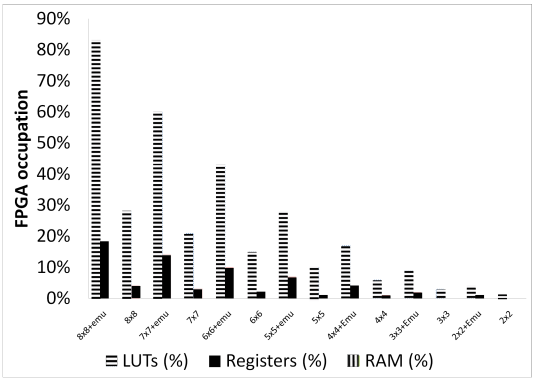
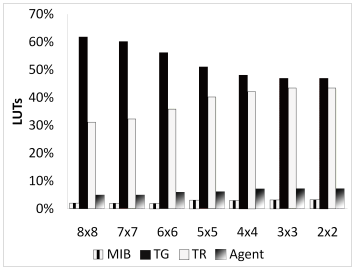


Fig. 6: FPGA occupation for synthetic traffic implementation

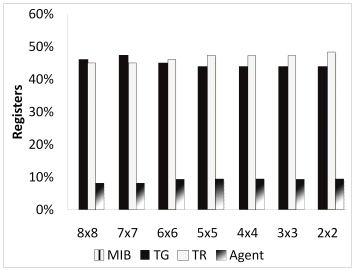
The additional FPGA resources used to implement the emulation platform are due to the SNMP components and the traffic model components. In order to quantify the influence of these components to the overall platform occupation, we analyze the resources occupation of only one emulation node. The Figure 7 depicts the LUTs and Registers occupation for the MIB, Agent, TR and TG components. We point out that the SNMP components (MIB and Agent) occupy only 7% of the LUTs and 8% of the Registers used to implement one emulation node. Indeed, most of the resources are applied to implement the traffic model components. The TG is the only component whose resources occupation significantly depends on the number of nodes implemented.

Henceforth, the resources occupation of the task graph traffic implementation is analysed. The Figure 8 depicts the FPGA resources occupation for different configurations of NoC and the emulation platform. We observe that for a given NoC size the emulation platform has 4 times more LUTs and 6 times more Registers than the implementation of only the NoC components. Furthermore, these implementations use RAM resources to implement the MIB registers. For this traffic model, the MIB holds $(20 * number\_of\_destinations + 36)$ 32-bit registers, by the other side, for the synthetic traffic model the MIB holds only $(7 * number\_of\_destinations)$ 32-bit registers. The number of destinations can be configured before the platform synthesis. In these experiments the number of destinations is equal to the number of nodes in the NoC, which is the worst case possible in relation to resources occupation.

In order to quantify the influence of the SNMP components to the overall platform occupation, we analyze the resources occupation of one emulation node. The Figure 9 depicts the LUTs and Registers occupation for the MIB, Agent, TR and TG components. The SNMP components represent only 15% - 30% of the LUTs and 6% of the Registers used to implement one emulation node. Indeed, most of the resources are applied



(a)



(b)

Fig. 7: LUTs (a) and Registers (b) occupation share for one emulation node of the synthetic traffic implementation

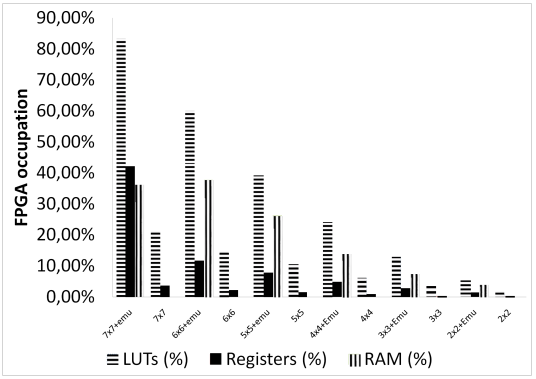
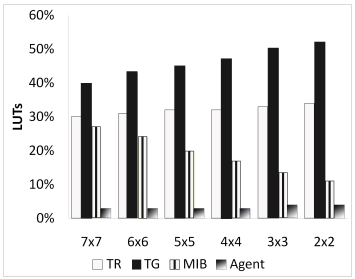


Fig. 8: FPGA occupation for task graph traffic implementation

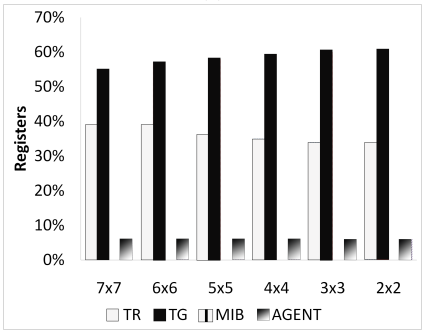
to implement the traffic model components. However the MIB has a growing occupation of LUTs which depends on the number of implemented nodes. These resources are used to implement the memory locations for the traffic model parameters. Usually, these memory components are implemented as RAM blocks in the FPGA. When the MIB is synthesized alone the synthesis tool implements them as LUTs. When the whole emulation platform is implemented, the MIB is implemented mostly as RAM blocks.

D. Results Discussion

The SNMP timing performance evaluation is presented in the subsections V-A and V-B. The execution of the SNMP operations are independent of the NoC size and these operations represent a very small fraction (0.004% - 1.7%) of the time spent on the execution of the traffic scenarios. Furthermore, the utilization of the differential MIB update reduces in 54% the amount of data exchanged between the FPGA and the host PC. Thus, it permits fast update of the MIB content. Although,



(a)



(b)

Fig. 9: LUTs (a) and Registers (b) occupation share for one emulation node of the task graph traffic implementation

the serial link between the FPGA and the host PC represents a communication bottleneck. A higher bandwidth link could allow faster MIB update and consequently faster emulation execution.

In the subsection V-C, it is presented the resources analysis for two different implementations of the proposed emulation platform. The results point out that the SNMP components represent a small fraction of the overall emulation platform. The MIB is the only SNMP component that depends on the complexity of the implemented traffic scenarios, because it must hold all the parameters related to the traffic model description. For the ML605 evaluation board, it is possible to implement the emulation platform for a 8x8 NoC (micro-benchmark scenarios) and a 7x7 NoC (task graph scenarios). The low resources occupation of the SNMP components is due to the lightweight circuits in charge to implement the SNMP functions.

The obtained results point out that the SNMP communication model is a fast and lightweight solution to drive a NoC emulation platform. Besides that, the circuitry needed to implement the SNMP components is quite simple.

VI. Conclusions

In this paper, we evaluate the SNMP protocol concepts to manage an FPGA-based NoC emulation platform. The implemented architecture enables the easy configuration of the emulation nodes, as well as it defines an interoperability model for the emulation components based on the MIB description. As a demonstration of the interoperability the proposed emulation platform integrates two distinct implementations of traffic models, which are defined by two distinct MIBs. The first one enables the creation of micro-benchmark synthetic traffic scenarios. The second one enables the creation of traffic scenarios based on tasks graphs. The experiments highlight that a light version of SNMP is very efficient for a light resources overhead.

Acknowledgment

Funding for this project was provided by a grant from la Région Rhône-Alpes as well as by the CNPq (process 245340/2012-2).

References

[1] Y. Ben-Itzhak et al. HNOCS: modular open-source simulator for heterogeneous NoCs. In *Int. Conf. on Emb. Comp. Systems*, pages 51–57, 2012.

[2] M. Coppola et al. OCCN: a network-on-chip modeling and simulation framework. pages 174–179. IEEE Comput. Soc, 2004.

[3] N. Genko et al. A novel approach for network on chip emulation. In *IEEE Int. Symp. on Circuits and Systems*, pages 2365–2368 Vol. 3, 2005.

[4] H. Hossain et al. Gpnocsim - a general purpose simulator for network-on-chip. In *Int. Conf. on Information and Communication Technology*, pages 254–257, 2007.

[5] Kouadri-Mostefaoui et al. Large scale on-chip networks : An accurate multi-FPGA emulation platform. In *11th EUROMICRO Conf. on Digital System Design Architectures, Methods and Tools*, pages 3–9, 2008.

[6] Y. Krasteva et al. A fast emulation-based NoC prototyping framework. In *Int. Conf. on Reconfigurable Computing and FPGAs*, pages 211–216, 2008.

[7] O. Laouamri and C. Aktouf. Enhancing testability of system on chips using network management protocols. In *Design, Automation and Test in Europe Conf. and Exhibition*, volume 2, pages 1370–1371 Vol.2, 2004.

[8] X. Li and O. Hammami. Multi-FPGA emulation of a 48-cores multiprocessor with NOC. In *Design and Test Workshop*, pages 205–208, 2008.

[9] S. Lotlikar, V. Pai, and P. Gratz. AcENoCs: a configurable HW/SW platform for FPGA accelerated NoC emulation. In *24th Int. Conf. on VLSI Design*, pages 147–152, 2011.

[10] F. Moraes et al. HERMES: an infrastructure for low area overhead packet-switching networks on chip. *Integr. VLSI J.*, 38(1):69–93, Oct. 2004.

[11] E. Pekkarinen et al. A set of traffic models for network-on-chip benchmarking. In *Int. Symp. on System on Chip*, pages 78–81, 2011.

[12] S. Prabhu et al. Ocin tsim- DVFS aware simulator for NoCs. In *Workshop on SoC Architecture, Accelerators and Workloads*, 2010.

[13] J. Tan, V. Fresse, and F. Rousseau. In *22nd IEEE Int. Symp. on Rapid System Prototyping*, pages 186–192, 2011.

[14] D. Wang, N. E. Jerger, and J. G. Steffan. DART: a programmable architecture for NoC simulation on FPGAs. In *Proc. of the Fifth ACM/IEEE Int. Symp. on NoC*, NOCS '11, pages 145–152, New York, NY, USA, 2011. ACM.

[15] P. Wolkotte, P. Holzenspies, and G. J. M. Smit. Fast, accurate and detailed NoC simulations. In *First Int. Symp. on Networks-on-Chip*, pages 323–332, 2007.

[16] R.P. Dick, D.L. Rhodes, and W. Wolf. TGFF: task graphs for free. In *Sixth Int. Work. on Hardware/Software Codesign*, pages 97–101, 1998.

[17] X. Li and O. Hammami. Multi-FPGA emulation of a 48-cores multiprocessor with NOC. In *Design and Test Workshop*, pages 205–208, 2008.

[18] System Management Bus. http://smbus.org/, [September 2014]

[19] Network Controller Sideband Interface (NC-SI) Specification. http://www.dmtf.org/sites/default/files/standards/documents/DSP0222_-1.0.1.pdf, [September 2014]