Formal Verification of Circuit-Switched Network on Chip (NoC) Architectures using SPIN

Anam Zaman
School of Electrical Engineering and Computer Science
National University of Sciences and Technology
(NUST), Islamabad, Pakistan
Email:anam.zaman@seecs.nust.edu.pk

Osman Hasan
School of Electrical Engineering and Computer Science
National University of Sciences and Technology
(NUST), Islamabad, Pakistan
Email:osman.hasan@seecs.nust.edu.pk

***Abstract*—Simulation techniques cannot cope with the distributive and reactive nature of Network on chip (NoC) architectures very well and thus compromise on the accuracy of the analysis results. Formal verification has been used to overcome these challenges but, to the best of our knowledge, has been mainly used for the verification of packet-switched NoC's. The main focus of this paper is on the formal verification of circuit-switched NoC's, which provide a dedicated channel for all communications with full bandwidth and thus are found to be more efficient than packet-switched NoCs in many contexts. In particular, the paper presents a generic methodology for the formal verification of circuit-switched NoC using the SPIN model checker. The proposed methodology provides generic modelling guidelines and identifies some properties, including deadlock freedom, starvation freedom, mutual exclusion and liveness, that are quite useful in the context of circuit-switched NoC. For illustration purposes, we use our methodology to verify the programmable NoC (PNoC) architecture, which is one of the most widely used circuit-switched NoC.**

I. Introduction

Traditional bus-based communication architecture is not able to pace up with contemporary System on chips (SoCs). It has become hard to deliver data from point to point during a single clock cycle. Moreover use of multiple buses increases design complexity and reduces scalability [15]. Data synchronization of on-chip modules via a global clock leads to high power consumption and electromagnetic interference (EMI) problems [3]. The concept of Network on chips (NoC) [7] integrates network theoretic concepts for on-chip communication between various processor cores of SoC architectures and offers a promising solution for the above-mentioned problems. It not only handles the synchronization issues but its modular nature allows it to be used with standard interfaces. Besides the scalability, NoC architectures are well-known to facilitate design-flow parallelism and reduce power consumption compared to the traditional bus based interconnects [15] [3] [8].

The communication network in a NoC consists of wires and routers [3] [7]. Memories, processors and other IP-block are connected to the router and the routing algorithm manages all the communication. The network layer of a NoC helps in transmitting data by using switching algorithms [23], which are categorized into packet and circuit based techniques. Packet-switched based NoC divides the data into packets which are transmitted to the dedicated node. Once received, data packets reassemble the sequence to recover the message. Under heavy data, such as video streaming, flow packets may get lost or sometimes get corrupted. Thus, packet switching needs a protocol for reliable data transfer. Circuit-switched, on the other hand, ensures reliable data transfer by defining a dedicated path between nodes. It guarantees full bandwidth and provides quality of service. The messages arrive in the same manner as they were transmitted. Circuit switched based NoC is ideal for voice based services [3] [15] and has been known to perform better than packet switched based NoC in many aspects [10] [21].

One of the main challenges in NoC architectures is their functional verification. Due to the distributed and reactive behaviors of NoC architectures, generating interesting test patterns for simulations is a major issue. This fact coupled with the non-exhaustive nature of simulation for analyzing large designs, makes a rigorous analysis of NoC architectures impossible. Especially when considering many-core systems, the number of different communication signals, grows exponentially with the number of cores. Even if some corner cases can be specifically targeted, there is no proof that these represent a worst-case scenario, and it is never possible to consider or even foresee all corner cases. Thus, simulation based analysis cannot be considered complete and often results in missing critical bugs, which is a very undesirable characteristic, given the safety and financial critical nature of the present-age many-core chips, where the NoC architectures are usually deployed [23] [7].

Formal verification methods [2] [26] have been used to overcome the above-mentioned limitations for many hardware designs. The main idea behind formal verification is to analyze the behavior of systems using mathematical reasoning or rigorous state-space exploration. The system implementation and specification are given to the formal verification tools and the goal is to identify corner cases in which the system model does not conform to its specifications.

Theorem proving [14] use deductive techniques for verifying the relationship between the logical specification and logical implementation of the given systems. The verification process may require explicit user guidance and thus can be quite tedious, especially when using more expressive higher-

978-1-4799-6890-9/14/$31.00 ©2014 IEEE

order logic [5]. Model checking [5] is the second mainstream formal verification method and involves the computer based mathematical modeling of the given system in the form of an automata or state-space. This model is then used within a computer to automatically verify that it meets rigorous specifications of intended behavior. Due to its mathematical nature of these formal verification methods, 100% completeness and soundness of the analysis can be guaranteed. Model checking seems to suit the context of NoC verification the best due to its automatic and hence user-friendly nature and the fact that the behavior of NoC architectures can be easily expressed in terms of a state-space. Moreover, the ability to provide counter examples in case of failures and the automatic nature of model checking makes it a more preferable choice for industrial usage as compared to the other interactive formal verification approaches like theorem proving.

Model checking has been successfully used for analyzing packet switched based NoC architectures (e.g. [23] [31] [11] [25]). However the main focus of these works is to verify particular aspects of NoC architectures in a very ad-hoc manner. Chen et all. [11] implemented the bi-network on chip (BiNoC) model in state graph manipulator (SGM). They checked four critical properties of the given NoC router, namely mutual exclusion, starvation freedom, deadlock freedom and conditions for traffic congestions. They successfully verified the mutual exclusion property but faced state-space explosion problem while verifying the deadlock-freedom and traffic congestion. On the other hand, starvation freedom property failed proving that Bi-NoC model is not a fair one. Hermes NoC was formally verified by Vinitha et all. [23]. Using the SPIN model checker, they verified reliable data transfer along with valid path selection by the router. Salaun et all. [25] also contributed to this field by verifying an asynchronous NoC. They used the model checking tool available in the CADP toolbox and verified the deadlock freedom along with correct sequencing of communications in a protocol. Holcomb [16] checked the Quality of Service (QoS) for the 2D-mesh NoC by verifying performance properties, such as latency bounds and synthesis of optimal buffer sizes, using the SAT based model checker ABC.

Theorem proving has also been widely used for NoC verification. Borrione [8] presented a formal meta-model for reasoning about network specifications, from the transport to the data link layer of the OSI architectural model. The model was instantiated on Octagon, Hermes and Spidergon NoC and verification flow was adopted by the ACL2 theorem prover. Another GeNoC was formalized by Broek [29] and is verified for both packet and circuit-switched based NoC in ACL2. They also verified that the 2D-mesh architecture [30], based on the Hermes NoC, behaves according to the specification. Moreover they extended their work [32] by adding two theorems to GeNoC, i.e, the network is deadlock free and can evacuate all injected messages.

Besides the main stream formal methods, like model checking and theorem proving, some other formal methods have also been used for the formal verification of NoC architectures. For example, EventB formalism is used to ascertain the architectural correctness of NoCs [4]. ForEVeR tool has been used to identify bugs in the NoC fabric [24]. Evaluation under self similar traffic is observed is [27]. Based on the above-mentioned literature review, it can be observed that most of the existing work on the formal verification of NoC is focused on packet-switched NoC. Similarly, none of these techniques provide a complete cross level verification of NoC and all of them use different sets of evaluation parameters for NoC verification.

This paper tends to overcome the above-mentioned limitations and proposes a generic methodology for the formal verification of any circuit switched NoC architecture. The main idea behind the proposed methodology is to use the SPIN model checker [17], which is an open source tool for the formal verification of distributed software systems. We can formally model or specify the behavior of NoC architectures in the PROcess MEta LAnguage (PROMELA) language. These models can then be verified to exhibit the desired functional properties using the SPIN model checker as it directly accepts PROMELA models. The main contribution of the paper lies in identifying the modeling techniques and abstractions that are tailored towards the circuit-switched based NoC architectures. Moreover, we also identify a set of properties that are of interest in the context of NoC and thus can be used with some adaptation to verify any NoC architecture.

In order to illustrate the utilization and effectiveness of the proposed methodology for the formal verification of real-world NoC architectures, we present the analysis of the Programmable Network on chip (PNoC) architecture [15], which a circuit-switched based NoC. It is a lightweight NoC architecture that is specifically designed for FPGA based systems and ensures data transfers with minimal overhead on a dedicated path. PNoC allows flexibility of having numerous router to node configurations along with parameterizable addresses. The main reason behind the choice of this case study is the above-mentioned strengths of this lightweight scheme compared to other traditional NoC techniques, like CLICHË [20], which consists of a fixed 2D mesh with one routing switch per computing node that makes it unsuitable for heavy data flow applications. Millberg also employed the same switching technique in their Nostrum NOC implementation [22].

The rest of the paper is organized as follows: Section II provides an overview of model checking and the SPIN model checker. The proposed methodology adopted for the verification of circuit-switched based NoC is discussed in Section III. Section IV presents the formal verification of the PNoC architecture. Finally Section V concludes the paper.

II. PRELIMINARIES

A. Model checking:

Model checking is a formal verification method for reactive systems. It is an algorithmic technique in which a state-based model of the given system is developed and its specifications are formulated in temporal logic. The model checker then automatically and exhaustively validates that the model is obeying the specifications. Model checking also facilitates debugging by providing error-trails [6] [17]. The verification is based on exhaustive state-space exploration and thus for large models the computation memory and time requirements grow exponentially, i.e, a problem that is usually referred to as the state-space explosion problem. A common remedy to this problem is to abstract away some of the uninteresting details of the model. Similarly, bounded and symbolic model checking techniques are also very helpful in coping with the state-space explosion problem [14]. Some of the widely used model checking tools are NuSMV, PRISM, CADP and SPIN. We have chosen the SPIN model checker for our work.

B. SPIN model checker:

The SPIN model checker is a general tool for verifying the correctness of asynchronous distributed software models in a rigorous and mostly automated fashion [17]. SPIN utilizes PROMELA (Programming Meta language), which offers communication and concurrency primitives inspired by the process algebras [19], as its formal specification language. PROMELA is primarily based on the Dijkstras guarded command language and has a syntax very much like C. The components of the given distributed system are modeled as processes, which are asynchronous in nature. It also supports multiple instances of a process. These processes communicate with each other via channels, which can be buffered or rendezvous. Variables can be declared globally or locally and can have different data types, such as int, bool, byte and short[6] [17]. Arrays and structures are also supported by SPIN. SPIN also supports random interactive guided simulation, which allows user to manually select every proceeding step in the simulation.

TABLE I
LTL Properties Notation

Propositional Connectives		Temporal Operations	
NOT	!	always	[ ]
and	&&	eventually	<>
or	||	until	u
implies	−>		
equivalence	<−>		

Verification properties are written in SPIN using the Linear Temporal Logic (LTL). Propositional connectives and temporal operators used in LTL properties are shown in Table 1 [10]. The properties are translated to Buchi automaton and then the SPIN built-in search algorithms, i.e., Depth First Search (DFS) or Breadth First Search (BFS) algorithm, search the state-space to check if that property holds for the given model or not. SPIN also has the capability of finding deadlocks or non-progress cycles. The state-space is expanded according to the DFS or BFS mode. Advance options not only allow the user to manually adjust memory size for state-space storage but also limit the state-space expansion.

Many model checking tools exist and each one excels in one or a set of application domains. For example, NuSMV [12] is known to provide excellent results for hardware verification and CBMC [13] and SLAM [28] are widely used for software verification. The SPIN [17] model checker caters for a wide range of applications, including both hardware and software systems and protocol verification. Many tools, such as the ABC model checker [9], support the verification of only synchronous. SPIN, on the other hand, supports both synchronous as well as asynchronous design verification. Moreover, some other unique characteristics of SPIN include the provision of the visual trail of errors, assertion based verification and the inbuilt properties checking mechanism for deadlocks and livelocks in the system. Finally, SPIN offers state-space reduction by using the partial order technique and the user is allowed to select between the exhaustive or partial space searching options. Thus, we found SPIN as most suitable tool for PNOC verification

III. Proposed Methodology

There are some general requirements that every NoC is expected to meet. Its architecture should be generic, scalable and able to cope up with faults. In terms of performance, it should exhibit small latency and guaranty throughput. The proposed formal verification methodology for circuit-switched NoC architectures, depicted in Figure 3, caters for all of these requirements and is primarily composed of the following steps:

A. Modeling Generic Circuit-Swicthed Based NoC in SPIN

The proposed methodology allows us to model generic circuit-switched NoC architectures, i.e., any NoC with arbitrary number of nodes and finite number of routers. Each node has a unique identity registered in the router with which it is connected. The router consists of a set of input and outputs, a controller and a communication switch. Router controller registers the address of all of its connected nodes. Moreover, it entertains the request of each node and schedules their access to utilize the switch. The communication switch or the switch box creates the communication path of the target and the master node.

The following steps allow us to model any NoC architecture in SPIN:

- **Identify process:** The first modeling step is to identify the processes in the given NoC architecture. We usually associate a process with every module, i.e., node, router and switch box of the given NoC. The behavior of every module is expressed in the process using its corresponding Finite-State Machine (FSM). Nodes of the circuit-switched NoCs do not require packetisation of the data with long overheads. Multiple nodes, can be modeled as

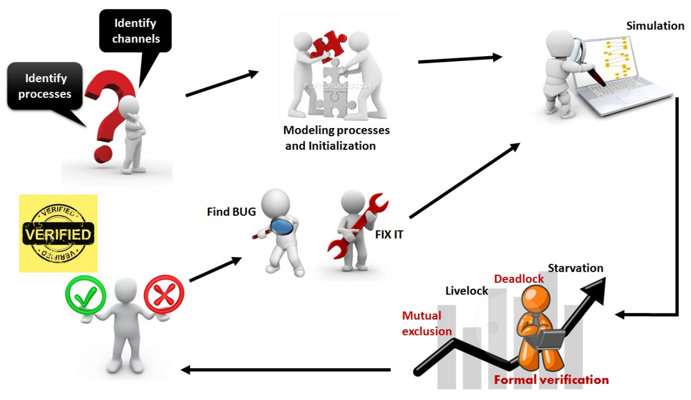

Fig. 1. **Proposed methodology**

an instance of the same process. All the communication within a process has to be defined via local variables.
To model a node of the circuit-switched NoC, the following points should be considered. It must start in an idle state. It must have data reading and writing capabilities. Moreover, it should be able to check the status flags of the communication link.
The router of the circuit-switched NoC also remains in the idle state until it receives a transmission request. It continuously listens the request signals to be able to respond in a timely manner. The router must be able to manage multiple requests.

- **Identify Input outputs:** Communication among various modules of the NoC is done via channels and thus each process needs to have its own input and output channels. Channels are responsible for delivering message among different processes. It is important to note that SPIN allows unidirectional channels only.

- **Initialization:** All the processes and inter module channels have to be initialized. The processes with *active* prefix allows them to be running in the initial system state but they have a limitation that they cannot accept any argument.

B. Functional Verification using Simulation

Once the model is ready in PROMELA, its complete FSM can be seen by the automaton viewer of the SPIN model checker for sanity check. Next we propose to check it via the random and interactive simulation methods of SPIN. The randomized test vectors often reveal some critical flaws in the bugs, which can be fixed by updating the PROMELA model. The main motivation of performing this simulation is to be able to catch PROMELA modeling flaws, that usually happen due to human errors before going to the rigorous and thus comparatively time consuming formal verification phase.

C. Formal Function Verification

As depicted in Figure 3, we propose to check four functional properties for NoCs, explained below, by model checking.

Deadlock Freedom: NoC architectures are quite susceptible to enter deadlocks, i.e., a situation under which two nodes are waiting for one another to transmit or receive data. Such deadlocks stall the entire communication and have to be absolutely avoided [11] [18]. Deadlocks in NoC are mainly introduced from the behavior of the router module. It is almost impossible to guarantee that there is no deadlock in a given router using simulation due to the huge number of possibilities that may cause deadlocks. Whereas, model checking can detect them automatically by checking a no-deadlock condition as an LTL property. Deadlocks may be reported due to a PROMELA modeling error or it could be a result of a system bug. Thus, in case of finding a deadlock, the corresponding error trail must be executed on the NoC model in PROMELA using simulation to identify the cause for addressing it.
Liveness: SPIN has the capability of verifying liveness properties without providing explicit LTL specifications. This property tracks non-progressing cycles in the system, which are associated with processes that do not halt in their non-critical section or infinitely trap in a non-ending loop.
Behavioral Properties: The behavioral properties of the given NoC architecture can be checked by formally specifying them in LTL. The translation from LTL to Buchi automaton is handled automatically by SPIN. In the proposed methodology for verifying circuit-switched NoC architectures, we call for verifying mutual exclusion and starvation freedom properties to check the behavior of the given NoC.

- ***Mutual exclusion***: This property ensures that hardware resources are utilized by one source at a time. It is very important to check this property in the context of NoC since multiple utilization of resources may result in effecting the reliable data transfer. Moreover, the data may not reach the specific target node. Finally it violates circuit-switch behavior, which only allows one node at a time to utilize its resource
- ***Starvation freedom***: This property ensures fairness of resource utilization such that no single node will share the resource for a long time. Holding a resource by a single node can lead other requesting nodes to be a continuous wait state, resulting in delayed data transfer.

Assertion based verification: SPIN supports assertion based verification as well and thus many interesting conditions can be checked during the verification process. For example, it is extremely undesirable if two nodes are granted access to share the datelines in circuit-switched based NoC and this scenario can be checked by using the following assertion:

```
Assert(grant1==1&&grant2==1)
```

It is a common occurrence to encounter state-space explosion problem during the verification phase. In this case, we propose to reduce the size of the model and thus the state-space by reducing the range granularity of variables or restricting the number nodes in the model.

The proposed methodology is general enough to be used to formally verify any circuit-switched NoC architecture. For illustrating its practical utilization and effectiveness, we use it in the next section for the formal verification of programmable NoC (PNoC) [15].

IV. Case Study: PNoC

A. *Programmable network on chip (PNoC):*

Programmable Network-On-Chip (PNOC) [15] is a circuit-switched NoC architecture and thus allows data transfers on a dedicated connection. Moreover, it requires no overhead for packetisation, packet header processing or packet buffering. This topology contains series of subnets where multiple nodes are connected to a single router through the router port interfaces. A light handshaking mechanism is required to establish or remove connection from a node. This flexible system allows runtime insertion and removal of nodes. PNoC has a small set of single bit controls signals and receiving (rx), transmitting (tx) signals. Figure 4 shows the connection between two nodes and a single router and more details about the signals can be found in [15]. The behavior of PNoC nodes, router, switch box and communication processes, which are the main components of any circuit-switched NoC architecture, is explained below;

Nodes are processing units which can send or receive data. A PNoC node has five control signals and a set of receiving (rx) and transmitting (tx) signals. It can *request* for service while specifying the target nodes address on the *tx_address* line.

Router is the core communication block as it controls the circuit switching among the nodes. It listens to node requests and then *grants* permission to the master node after consulting its routing table. In case when multiple nodes request simultaneously, it non-deterministically grants access to any one of them. Moreover, it also notifies the slave node to establish connection. Once the connection is established between two nodes then they can freely share data. If the desired node or hardware resource (switch box) is busy in entertaining other services, the router sends a *pend* signal to the requesting node. The router also de-asserts the *pend* signal after the resource node becomes free. Moreover, it also listens to the *release* signal from the node incase the node wants to terminate the connection.

Switch box is the unit that physically connects all the appropriate *rx* and *tx* signals of two nodes.

To establish connection between nodes and router, a simple handshaking mechanism is performed. Master node sends

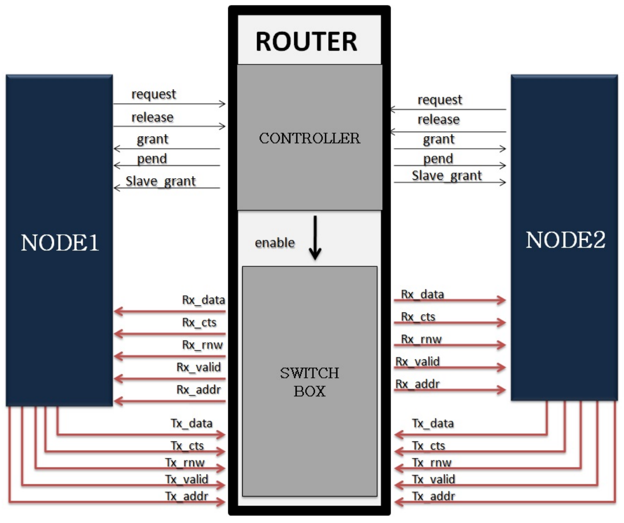

Fig. 2. **PNoC architecture**

request to the router along with the address of target node. Router consults its routing table and checks whether the communication link and slave node is free for data communication. Once approved, it sends *grant* to master and informs the slave node via the *slave_grant* signal. Moreover, it connects *rx* and *tx* lines of the master and slave nodes via a switch box. In the write mode, the master node sets *tx_valid=1* and *tx_rnw=0*, and then captures the returned data from the slave node on rx_data lines upon receiving *rx_valid=1*. In the read mode, master nodes set *tx_valid=1* and *tx_rnw=1*. If the slave wants to suspend the data transfer then it can do so by deasserting its *cts* signal.

Once the communication link is established, data is freely shared between nodes and on completion, the node sends a release signal to the router, which ends the communication.

B. *Formal Verification of PNOC in SPIN*

1) Modeling of PNoC in PROMELA

The PROMELA modeling of PNoC mainly involves three modules, i.e., nodes, router and switch box. We have modeled and analyzed the PNoC for three different configurations i.e., one router with two, four and eight nodes. Nodes are generic in nature and thus have been instantiated multiple times. The pseudo codes of these models along with some description is given below:

PNoC Node:

The node module in PNoC can initiate requests to the router to communicate with other nodes. The node that generates the request is termed as a master node and targeting node is termed as a slave node. The PROMELA model of a PNoC node is given in Pseudo code 1.

A node is initialized to be in the idle mode. Every node can initiate a request to the router with the address of the target node. This behavior is modeled non-deterministically, such that the request generation and the target node-address are both chosen in a non-deterministic fashion. In case of request generation, the node move to the receiving mode, where it waits for either *grant*, *pend* or *slave _grant* signal. On receiving the *grant* signal, it moves to the *node_write_mode* (if the node wants to transmit data) where it sends a set of communication signals. Once the data is being transferred, the node moves to the *node_windup* mode where it sends the *release* signal to the router; thus requesting end of data transfer. Similarly, if the node wants to read data from other nodes on receiving grant, it moves to the *node_read_mode*, where by using a set of communication signals, it receives data and upon completion, it also sends the release signal to the router, thus demanding end of connection.

When node has to act as a slave upon receiving *slave grant*, it moves to the *slave* mode and acts according to the request of read or write from master node. If the hardware resource, such as switch is used by some other node then the router sends a *pend* signal to the node under consideration, so that it waits until the *pend* is over and again requests for communication.

Pseudo code 1: Node

```
node_idle:
//choose non-deterministically
if
::request!1->tx_addr!addr->write=1/write=0->goto
node_receving
::request!0->goto node_idle;
node_receving:
:: grant? Grant->if grant==1->
if write==1->goto node_write_mode
if write==0->goto node_read_mode
:: Pend?pend->if pend==1->goto pend mode
:: Slave_grant?sl_grant->if sl_grant==1->goto
slave mode
node_write_mode:
::initialize handshaking mechanism
::send data-> goto node_windup// transmit data
as far as tx_cts=1
node_read_mode:
::rx_cts?rx_cts->
if rx_cts==1->
::rx_data?data// receive data as far as
rx_cts==1
goto node_windup
else goto node_windup
pend mode:
::pend?pend->
if pend==1/*wait till pend=0*/request!0;
if pend==0->request!1;
slave mode:
if rx_valid==1 & rx_rnw==0-> goto node read
mode
if rx_valid==1 & rx_rnw==1-> goto node write
mode
node windup:
Release! 1->request! 0->goto node_idle
```

PNoC Router:
The behavior of the router is given in Pseudo code 2. It receives the requests from nodes and consults its routing table to check the availability of the targeting node's address. Once the switch becomes free, then the router sends a grant signal to the master and slave nodes. Once the connection is established, the router starts to wait for their release signal. Upon receiving the release signal from both of the communicating nodes, it again moves to the router idle mode, where it waits for further requests of nodes.

Pseudo code 2:Router

```
router idle:
:: Request[m]?Request[m]->goto routingtable
routingtable:
rx_addr?rx_addr->s=rx_addr;
if switch_enable==0
:: grant[m]!1->sl_grant[n]=1->enable_switch!1->
goto wait release mode
if Switch_enable==1
::pend[m]!1 (wait until switch
enable=0)->pend[m]!0->
goto router idle
release mode:
Release[m]? release_m
Release[s]? release_s
if
release_m==1&release_s==1->goto router idle;
else goto release mode;
```

Pseudo code of switch box:
The switch box, mentioned in Pseudo code 3, is controlled by router. All the tx lines are connected to the rx lines of the switch box and vice versa. The switch box is usually in *switch_idle* state. If router enable the switch box then it connects its rx lines with tx lines. Thus, corresponding nodes send their data on these lines. Once data is transferred, the router disables the switch box and disconnects the connection. The switch box again moves to its idle position.

Pseudo code 3: Switch Box

```
switch idle:
::enable?enb;
If enb==0->goto switch idle;
If enb==1->
::rx_valid connects tx_valid
::rx_rnw connects tx_rnw
::rx_cts connects tx_cts
::rx_data connects tx_data
::rx_addr connects tx_addr->
goto switch idle
```

Initialization process:
An initialization process is used to initialize the router, nodes, all of the channels involved in networking and switch box. At first all the global channels are initialized, then the nodes and the router are initialized. For example, for the case of 8 nodes, the code for the node module would be instantiated 8 times.

2) Simulation of PNoC
After modeling the PNoC architecture in SPIN, simulation is the first check that evaluates its functionality. PNoC modules are modeled in a non-deterministic way. At first, guided-simulation is performed in order to ensure that all the signals are received properly. Random simulation allows the nodes

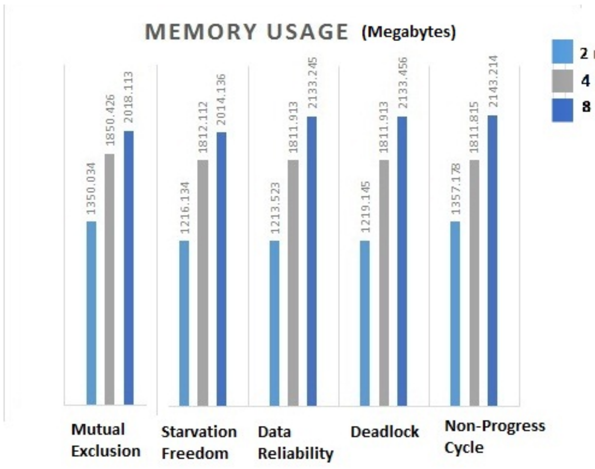

Fig. 3. **Memory utilization for different node**

to behave nondeterministically, a node can randomly send request and address of any other slave node. Any of the 8 nodes can send a request of communication with any other node connected to router. Router listens to these requests and acts accordingly. The track of this randomness can visually be seen via simulation based trails. In this way many unpredicted errors or bugs can be found.

3) *Formal verification of PNoC*

As described in our methodology, we checked the liveness and deadlock freedom of our PROMELA model of PNOC and no deadlock and non-progress states were found.

We also checked the mutual exclusion property to ensure that that hardware resources are utilized by one source at a time. When a PNoC's router sends a grant signal to a node then this particular node should be the only one to use the respective resources. We checked this behavior by using the zero-one-hot relation for all grant signals of all the nodes. In LTL format it can be written as follows:

```
LTL p0 [] (!(grant1 & grant2 & grant3 &
         grant4 & ......grantn));
```

The *grant* signal is asserted only when the node is given the authority to read or write as a master. Thus, in order to ensure mutual exclusion only one of the grant signals can be 1. This property was verified for our model of the PNoC, which ensures the mutual exclusions holds for every possible configuration of our model, something that cannot be ascertained via simulation very easily.

The starvation freedom of our model was checked by ensuring the fairness of resource utilization, i.e., no single node should occupy a resource for a long time. In the case of PNoC, resources are allocated by giving the grant=1 signal to the nodes. Thus, we checked the starvation property by observing the priority array of the router, which stores the process ID of the master nodes. In order to ensure starvation freedom in the PNoC architecture, there should be no continuous repetition of the same process ID in the priority array. This behavior can be expressed as a LTL property as follows:

```
                LTLp5
[]((grant[n]==1)-><>(Release[n]==1));
```

This property states that if any node is granted 1 than eventually in the future it will assert its release signal. In other words, it ensures that no node is granted 1 forever.

Fairness is also checked by the array method. The node that is given the grant stores its ID in the get_id function, which is basically defines an array of node IDs. Thus, the fairness can be ensured if no two consecutive entries in this array are same using the following assertion:

```
Id_one=Array[i]; Id_two=Array[i+1];
      Assert(Id_one==Id_two);
```

We also checked the data reliability of the PNoC architecture by ensuring that the data transmitted from one node is accurately received at the target node. The limitation for checking this behavior is that it can be checked only after complete data reception. Therefore, the assertion based verification is the only way to check this behavior since model checking using LTL properties cannot be used to verify it. This behavior can be checked by the following assertion:

```
Assert (rx_data!=tx_data)
```

This assertion would raise a flag if the transmitted data is not found to be equal to the received data at any point in time of the system execution. This property becomes true only when the data reception is complete at the slave node, which is consistent with the expected behavior as well.

For the above-mentioned verification of PNoC, a Quad core 2.5 GHz processor running windows 7 professional, SPIN version 6.2.3 along with the ispin version 1.1.0. was used. The PNoC verification is done for a maximum of eight nodes and a single router, which comprises of 11 processes and 500 lines of code.

The verification was done for the exhaustive storage mode along with the option of working up to maximum depth of 257269. The detail of resource utilization and verification time of each property verified is shown in Figures 5 and 6, respectively, for different number of nodes.

V. CONCLUSIONS

This paper presents a formal verification methodology for circuit-switched based NoC architecture. The proposed method mainly utilizes the SPIN model checker to verify the functional properties and quality of service of the given NoC. To the best of our knowledge, this is the first model checking approach for verifying circuit-switched based NoC. For illustration purposes, the paper presents the formal verification of PNoC architecture, which is circuit-switched based flexible NoC. This case study clearly indicates the applicability of the proposed methods to verify other circuit-switched PNoC architectures. We plan to apply our methodology to other circuit-switched PNoC architectures presented in [15]. Moreover, another interesting future direction is to look at the performance of various circuit-switch PNoC architectures using a probabilistic model checker, like PRISM [1].

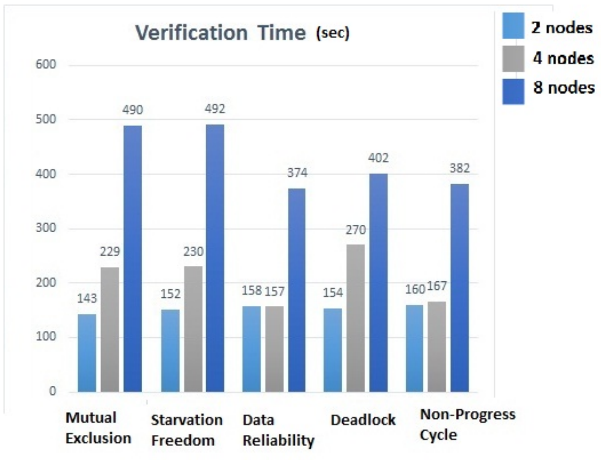

Fig. 4. **Verification Results**

References

[1] G. Norman A. Hinton, M. Kwiatkowska and D. Parker. Prism: A tool for automatic verification of probabilistic systems. pages 441–444. International Conference on Tools and Algorithms for the Construction and Analysis of Systems, 2006.

[2] J. Abria. Formal methods: Theory becoming practice. *Journal of Universal Computer Science, vol. 13, no. 5*, 2007.

[3] A. Agarwal, C. Iskander, and R. Shankar. Survey of network on chip (noc) architectures & contributions. Number 1, pages 13–27. Journal of Engineering, Computing and Architecture, vol. 3, 2009.

[4] M. Andriamiarina, H. Daoud, M. Belarbi, D. Mry, and C. Tanougast. Formal verification of fault tolerant noc-based architecture. First International Workshop on Mathematics and Computer Science, 2012.

[5] C. Baier and J.P. Katoen. Principles of model checking. 2008.

[6] M. Ben-Ari. *Principles of the Spin Model Checker*. 2008.

[7] L. Benini and D. Bertozzi. Xpipes: A network-on-chip architecture for gigascale systems-on-chip. pages 18–31. IEEE Circuits and Systems Magazine, vol. 4, no. 2, 2004.

[8] D. Borrione, A. Helmy, L. Pierre, and J. Schmaltz. Formal approach to the verification of networks on chip. EURASIP J. Embedded Systems, 2009.

[9] R. K. Brayton and A. Mishchenko. Abc: An academic industrial-strength verification tool. volume 54, pages 24–40. Proceeding of the 22nd International Conference on Computer Aided Verification, Springer, July 2010.

[10] K.-C. Chang, J.-S. Shen, and T.-F. Chen. Evaluation and design trade-offs between circuit-switched and packet-switched nocs for application-specific socs. pages 143–148. Design Automation Conference, 2006.

[11] Y. Chen, W. Su, P. Hsiung, Y. Cherng Lan, Yu. Hu, and S. Chen. Formal modeling and verification for network-on-chip. pages 299–304. In Green Circuits and Systems, 2010.

[12] A. Cimatti, E.M. Clarke, F. Giunchiglia, and M. Roveri. NuSMV*: a new Symbolic Model Verifier*. Number 1633 in Lecture Notes in Computer Science. Springer, Trento, Italy, July 1999.

[13] Edmund Clarke and Daniel Kroening. Hardware verification using ANSI-C programs as a reference. In *Proceedings of ASP-DAC 2003*, pages 308–311. IEEE Computer Society Press, January 2003.

[14] J. Harrison. *Handbook of Practical Logic and Automated Reasoning*. Cambridge University Press, 2009.

[15] C. Hilton and B. E. Nelson. Pnoc: a flexible circuit-switched noc for fpga-based systems. pages 181–188. Computers and Digital Techniques, IEE Proceedings, vol. 153 (3), 2006.

[16] D. E. Holcomb. Formal verification and synthesis for quality-of-service in on-chip networks. Master's thesis, University of California, Berkeley, 2013.

[17] G. J. Holzmann. The model checker spin. pages 279–295. IEEE Transactions on Software Engineering,Volume 23 Issue 5, 1997.

[18] L. Ribeiro F. Wagner J. Otero, F. Dillenburg. Formal verification of deadlock recovering algorithm of heterogeneous noc routing support.

[19] R. Kazhamiakin, M. Pistore, and M. Roveri. Formal verification of requirements using spin: A case study on web services. pages 406–414. Conf. Software Eng. and Formal Methods, 2004.

[20] S. Kumar and Jantsch. A network on chip architecture and design methodology. pages 105–112. IEEE Computer Society Annual Symposium on VLSI, 2002.

[21] J. Liang, A. Laffely, S. Srinivasan, and R. Tessier. An architecture and compiler for scalable on-chip communication. volume 12, pages 711–726. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2004.

[22] R. Thid S. Kumar M. Millberg, E. Nilsson and A. Jantsch. The nostrum backbone - a communication protocol stack for networks on chip. pages 693–696. IEEE International Conference on VLSI Design, 2004.

[23] V.A. Palaniveloo and A. Sowmya. Application of formal methods for system-level verification of network on chip. IEEE Computer Society Annual Symposium on VLSI, 2011.

[24] R. Parikh and V. Bertacco. Forever: A complementary formal and runtime verification approach to correct noc functionality. volume 13. ACM Transactions on Embedded Computing Systems, 2014.

[25] G. Salaun, W. Serwe, Y. Thonnart, and P. Vivet. Formal verification of chp specifications with cadp illustration on an asynchronous network-on-chip. pages 73–82. International Symposium on Asynchronous Circuits and Systems, 2007.

[26] Z. Sharifi, S. Mohammadi, and M. Sirjani. Comparison of noc routing algorithms using formal methods. pages 385–410. International Conference on Parallel and Distributed Processing Techniques and Applications, 2013.

[27] S.kundu, K. Manna, S. Gupta, K. Kumar, R. Parikh, and S. Chattopadhyay. A comparative performance evaluation of network-on-chip architectures under self-similar traffic. volume 1, pages 163–182. International Conference on Advances in Recent Technologies in Communication and Computing, 2009.

[28] V. Levin T. Ball and S. K. Rajamani. A decade of software model checking with slam. volume 54, page 6876. Communications of the ACM, ACM, July 2011.

[29] T. van den Broek and J. Schmaltz. A generic implementation model for the verification of networks-on-chips. pages 130–134. Workshop on the ACL2 Theorem Prover and Its Application, Northeastern Univ., Boston MA, USA, 2009.

[30] T. van den Broek and J. Schmaltz. Towards formally verified networks-on-chips. pages 184–187. Proceedings of Formal Methods in Computer Aided Design, 2009.

[31] B. Venu and A. Singh. Formal verification methodology consideration for network on chips. pages 220–225. International Conference on Advances in Computing, Communications and Informatics, 2012.

[32] F. Verbeek and J. Schmaltz. Formal specification of networks-on-chips: Deadlock and evacuation. pages 1701–1706. Design, Automation and Test in Europe, 2010.