Electronic Communications of the EASST
Volume X (2013)

Proceedings of the Automated Verification of Critical Systems (AVoCS 2013)

Functional and Performance Analysis of Network-on-Chips Using Actor-based Modeling and Formal Verification

Zeinab Sharifi, Mahdi Mosaffa, Siamak Mohammadi and Marjan Sirjani

16 pages

Guest Editors: Steve Schneider, Helen Treharne
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Functional and Performance Analysis of Network-on-Chips Using Actor-based Modeling and Formal Verification

Zeinab Sharifi[1], **Mahdi Mosaffa**[1], **Siamak Mohammadi**[1] **and Marjan Sirjani**[2*]

[1] z.sharifi@ut.ac.ir, m.mosaffa@ut.ac.ir, smohammadi@ece.ut.ac.ir
Department of Electrical and Computer Engineering,
Tehran University, Tehran, Iran

[2] marjan@ru.is
School of Computer Science,
Reykjavik University, Reykjavik, Iceland

Abstract: Network on Chip (NoC) has emerged as a promising architecture paradigm for todays many-core systems. As complexity grows in NoCs, functional verification and performance prediction in the early stages of the design process are suggested as ways to reduce the fabrication cost. Formal methods have gained more attention as alternative ways for analyzing NoC designs. In this paper we propose a method to model different characteristics of the system, and also verify various functional and performance properties by generating the full state space of the model for different scenarios. We present a formal model for two-dimensional mesh Globally Asynchronous Locally Synchronous (GALS) NoCs with four-phase handshake communication protocol, using the actor-based modeling language Rebeca. Functional and timing behaviors, routing algorithm and communication protocol are captured in the model. Deadlock freedom, message arrival, and end-to-end packet latency are checked. In order to analyze large NoCs we propose a scalable approach based on compositional verification for estimating maximum end-to-end packet latency. The compositional approach is specific for the XY-routing algorithm. Results of verification are compared and matched to simulation results of HSPICE using 32nm technology.

Keywords: Network on Chip (NoC), Rebeca, Compositional verification, Actor model

1 Introduction

Through technology shrinkage, multiprocessor systems on chips (MPSOCs) have emerged as viable solution to the growing complexity. Network on chip (NoC) is a promising interconnection paradigm for these systems. Fully synchronous design of NoCs faces some problems. Thus, Globally Asynchronous Locally Synchronous (GALS) NoC [ITR11] has gained attention in designing of such systems. However, GALS-based NoCs encounter two main challenges: (1)

[*] The work of this author has been partially supported by the project Timed Asynchronous Reactive Objects in Distributed Systems: TARO (nr. 110020021) of the Icelandic Research Fund.

functional verification, to check if the desired properties are met, and (2) performance evaluation in various stages of the design process to choose the proper design parameters.

As fabrication cost is high, it is desirable to tackle the two challenges before having the first prototype and even in the early stages of design process. To do so, it is required to have a model of the system with proper details, to perform model -based analysis. Moreover, using a technique that allows for high abstraction in the model enables the designer to perform a series of analysis in various stages of design process. However, to the best of our knowledge existing works do not present a suitable model for a GALS NoC with enough detail to be verified against both functional and performance properties.

This paper uses model checking to confront both challenges simultaneously and make use of one model for verifying both functional and performance properties. A formal model for GALS NoC is presented that can be used for estimating end-to-end packet latency, as well as checking functional properties like deadlock freedom and successful arrival of packets to their destination. Functional and timing behaviors, communication protocol, routing and scheduling algorithms are considered in the model. Based on the properties of interest, the model can be extended to support more details of the system.

One important point in asynchronous systems is that the lack of a reference clock leads to an interleaved execution of processes. Therefore, in GALS NoCs, a sent packet might be delayed by different number of disrupting packets and may have various end-to-end latencies. Thus, timing analysis in these systems is required to enable making suitable design decisions to avoid deadline miss for packets travelling through the network. For analysis of such systems it is essential to consider all possible behaviors of the system and generate the whole state space. However, existing work based on simulation techniques cannot be applied for exhaustive verification. Also, ensuring correctness to a certain degree using simulation is highly time-consuming. Formal methods and more specifically model checking are alternative approaches that can be used for both performance evaluation and correctness checking and allow us to perform exhaustive search in the state space [BHHK10].

Using an actor-based [Hew72] model with formal verification support allows us to model the asynchronous behavior of GALS NoC naturally. Here, Timed Rebeca (Reactive Objects Language) [SAB04, KKK$^+$12, SJ11] is used as the modeling language. Timed Rebeca is an actor-based modeling language capable of modeling functionalities and timing behaviors of asynchronous systems. In an actor model there are numbers of actors which are communicating via message passing. Similarities between the computational model of Rebeca and GALS NoC, leads us to a natural and easy to understand model.

Due to the asynchronous communication, applying an exhaustive verification on large NoCs may result in state space explosion. To alleviate this problem we present a method based on compositional verification. The method computes the maximum end-to-end latency in GALS NoCs with XY routing algorithm in two steps. It breaks the path of a packet to its destination into horizontal and vertical sub-paths and then performs latency estimation in each sub-path separately. At the end, the results for each sub-path are combined to get latency estimation of the whole path.

As an example of a NoC, we model and analyze ASPIN (Asynchronous Scalable Packet-switching Integrated Network) [SGM08], which is a fully asynchronous two-dimensional GALS NoC design using XY routing algorithm. Our experimental results are compared to results of simulations for a 4×4 ASPIN. Comparisons show that our results match the results of simulations. In another example we performed some experiments to show how our method can be used in making design decisions for memory location based on performance evaluation.

A general comparison between alternative analysis approaches, i.e. simulation and classical analytical performance analysis, is given in [BHHK10]. To be more specific, comparing to the works based on simulation this model can be used for exhaustive verification of different sizes of NoC in the early stages of design process. Comparing to the works based on mathematical and analytical analysis, this model considers hardware features including buffers and link delays, buffer status and also communication protocol, and captures the asynchronous communication paradigm. For a comparison with other formal methods that are applied for analyzing NoCs we shall emphasize both on the modeling and model checking techniques. Our approach has the advantage of using an actor-based model that due to its asynchronous computation model makes modeling natural and flexible. Also, the verification tools can efficiently reduce the state space that need to be checked, because specific actor-based reduction techniques are implemented within the tools [SAB04, KKK$^+$12]. The novelty of the present work is proposing a formal verification approach based on actors for modeling and analyzing GALS NOC. Using Timed Rebeca and the supporting tools we showed how GALS NOC can naturally be modeled and how functional correctness, performance measures, and design problems can be easily formulated. Our experiments show the validity, efficiency and scalability of our approach. In summary, the contributions of this paper are:

1. Proposing a user-friendly and flexible method for actor-based modeling of GALS NoC using Timed Rebeca where sufficient details can be added to the model based on the properties to be verified.

2. Presenting a method for using formal verification tools of Rebeca for checking functional and performance properties of NoCs which based on experiments has shown to work efficiently.

3. Using ASPIN to show our modeling and verification approach and comparing the results with simulation-based experiments as a validity check for our approach.

4. Introducing a more scalable method based on compositional verification to predict the maximum end-to-end latency of a sent packet specific for XY-routing.

The remainder of the paper is organized as follows. Section 2 contains related works. In Section 3 preliminaries are introduced, and Section 4 presents ASPINs model in Rebeca. The analysis using model checking as well as our compositional verification method is explained in Section 5. Results are shown in Section 6 and the conclusion is presented in Section 7.

2 Related Work

There exists several works based on simulation for analyzing on-chip communications. Hardware simulators such as Nirgam [NIR] and gem5 [Gem11] can be used for analyzing NoCs. However, with these tools it is not possible to model the details of GALS NoC functionality with precise timing information. To the best of our knowledge none of the existing simulation-based tools can model and analyze asynchronous behaviours in these systems, since they are non-exhaustive and cannot check all possible behaviours in asynchronous systems.

In various similar works, formal methods have been applied to simultaneously overcome the two specified challenges of functional verification and performance evaluation. An excellent overview on joining forces of model checking and performance evaluation is presented in [BHHK10]. In [Mad09] deadlock detection for applications in MPSoC is performed, and a timed automata model is used for performance evaluation. A Petri net model is presented in [NB12] for performance evaluation of asynchronous circuits. In [CSH$^+$10] functional behavior of a NoC is modeled in Extended Timed Automata, and its router is verified against some functional properties. A refinement based approach is applied to model a 3-dimensional NoC in [KPSD11]. Timing behavior and also hardware parameters like write and read delays for buffer and link delay are not considered in the model. In none of the above works GALS NoC was analyzed.

The first work for automatic synthesis of GALS-based systems has been done in 2008 at Newcastle University [Das08]. In this work the Signal Transition Graph (STG) and System Transition Synchronous Label are used. However, these models are not suitable for modeling large systems, since their low level nature makes model of large systems very complicated. Also, the computational model is not proper for performance evaluation. In [GAE11] Manchester and Newcastle Universities defined the GAELS project in 2011. In this project they want to model components of GALS NoC with elastic circuits using Petri net. In addition, in [Bul10, MS10] Interactive Markov Chain (IMC) and Interactive Probabilistic Chain (IPC) have been used for performance evaluation of such systems. The models are only used for performance evaluation, and functional verification was not considered in these works. Moreover, IMC and IPC are used in [CHLS10] to model a buffer used in NoC design. However, details of hardware timing and link model are not mentioned. In [FTHJ09] an analytical method based on Markov chain stochastic processes is proposed for computation of mean latency of the end-to-end communications via a 2-dimensional mesh NoC. The method approximates the latency of crossing each core in the path, assuming probabilities for the number of existing disrupting packets that want to pass from the same output port. Using probabilities reduces the state space at the expense of losing the buffer analysis.

In this paper, we use formal methods to perform functional verification and performance estimation on GALS NoC on the same model. In contrast to existing works based on formal methods, our model considers hardware details like link and buffer (read and write) delays. Also, the model could be easily extended to contain more details in various stages of design flow and can help the designer to make better architectural choices.

3 Preliminaries

3.1 ASPIN

ASPIN is a fully asynchronous two-dimensional mesh NoC with physically distributed routers in each core. ASPIN uses the storage strategy of input buffering, and each input port is provided by an independent FIFO buffer. Packets arrived from different sides (from neighboring routers on four sides and the local core), are stored in the respective FIFO buffer. If there is more than one request for an output port, a round robin policy is used for the arbitration.

ASPIN uses XY routing algorithm to route packets from input ports to output ports. Using this algorithm, packets can only move along X direction first, and then along Y direction to reach their destination. Communications between routers are established using four-phase handshake protocol. The protocol uses two signals namely Req and Ack. To transfer a packet, first, sender sends a request by rising Req signal, and waits for an acknowledgment from the receiver. All signals return to zero before the next packet could be sent[SF01].

3.2 Timed Rebeca

Rebeca is an actor-based modeling language with a Java-like syntax. Actors can be considered as a reference model for concurrent computation. A Rebeca model consists of reactive classes and a main part that contains instantiation of reactive objects (rebecs) from reactive classes. Rebecs have encapsulated states and their own execution thread. Each rebec contains a set of state variables, methods and a set of known rebecs with which it can communicate. States of a rebec consists of the valuation of state variables. Communication is through asynchronous message passing. Sending a message to a rebec will cause the invocation of corresponding message server. Each rebec has an initialization method (like a constructor) with the same name as the rebec which is executed while instantiating of the rebec. Parameters of this method are used for initializing the rebec and its known-rebecs. Each rebec has a buffer, called a queue, for arriving messages. The queues scheduling policy is FIFO. In each step a rebec is executed by removing a message from the top of its queue and executing its corresponding message server. The execution of a message server is an atomic execution of its body and may not be interleaved with other message servers.

Timed Rebeca is an extension to Rebeca, capable of modeling timing and functional behaviors of distributed reactive systems. In a Timed Rebeca model each rebec has its own local clock. The local clocks can be considered as synchronized distributed clocks. The execution of message servers is still atomic and can lead to progress of time in that rebec. To model timing behaviors of a system like computation time, message delivery, message expiration, and period of event occurrence the three below constructs are provided:

1. **delay (t):** this construct causes a delay of t time units.

2. **after (t):** this construct is paired with an invocation of a message server (method call), and causes a message to be sent with a delay of t units of time (it does not cause a delay for the caller rebec).

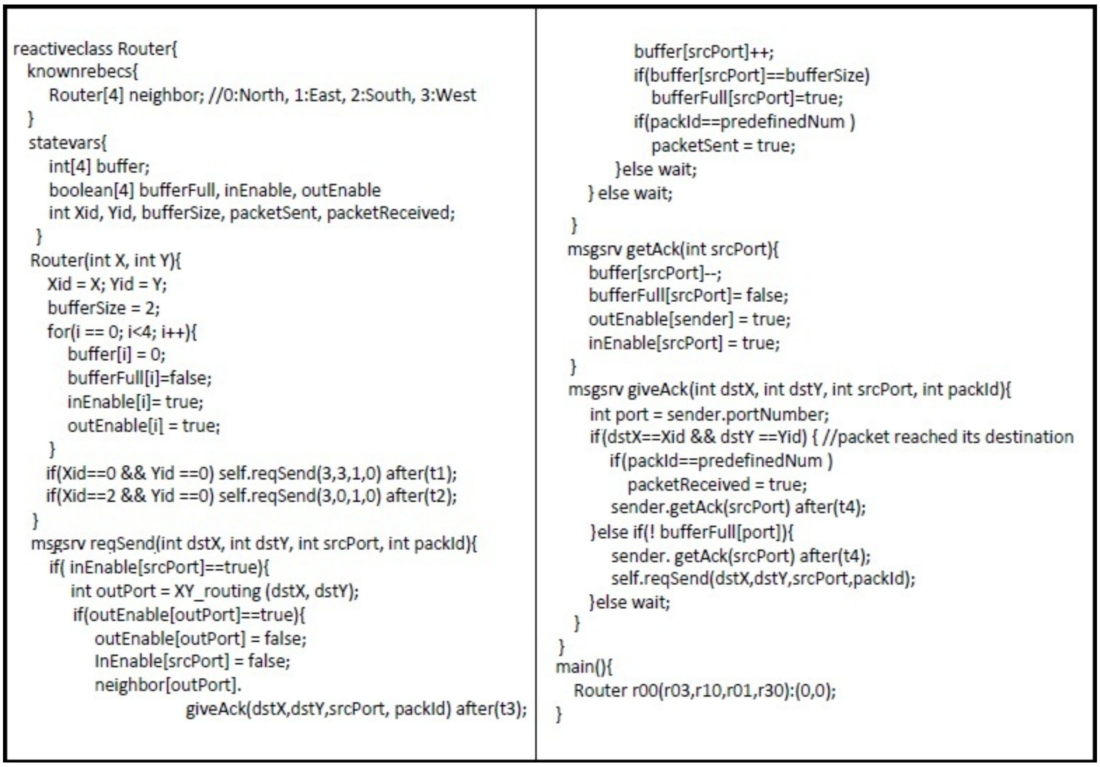
3. **deadline (t):** this construct is paired with an invocation of a message server (method call), and the corresponding message will be purged from the queue after t time units.

4 Modeling NoC Using Timed Rebeca

To use model checking in our analysis, first, the system should be modeled using Timed Rebeca. In this section, we present ASPIN model in Rebeca as a benchmark to be analyzed in Section 5. In this work, we check two functional and one performance properties of GALS NoC:

- Deadlock freedom (in an application with specified traffic pattern)
- Successful arrival of each packet at its destination
- Estimation of maximum end-to-end packet latency

Focusing on the above properties we need to consider the topology of the communication, routing algorithm, buffer status, and communication protocol in the model, we also have to capture



```
reactiveclass Router{
  knownrebecs{
     Router[4] neighbor; //0:North, 1:East, 2:South, 3:West
  }
  statevars{
     int[4] buffer;
     boolean[4] bufferFull, inEnable, outEnable
     int Xid, Yid, bufferSize, packetSent, packetReceived;
   }
  Router(int X, int Y){
     Xid = X; Yid = Y;
     bufferSize = 2;
     for(i == 0; i<4; i++){
        buffer[i] = 0;
        bufferFull[i]=false;
        inEnable[i]= true;
        outEnable[i] = true;
     }
     if(Xid==0 && Yid ==0) self.reqSend(3,3,1,0) after(t1);
     if(Xid==2 && Yid ==0) self.reqSend(3,0,1,0) after(t2);
  }
  msgsrv reqSend(int dstX, int dstY, int srcPort, int packId){
     if( inEnable[srcPort]==true){
        int outPort = XY_routing (dstX, dstY);
        if(outEnable[outPort]==true){
           outEnable[outPort] = false;
           InEnable[srcPort] = false;
           neighbor[outPort].
                       giveAck(dstX,dstY,srcPort, packId) after(t3);
           buffer[srcPort]++;
           if(buffer[srcPort]==bufferSize)
              bufferFull[srcPort]=true;
           if(packId==predefinedNum )
              packetSent = true;
        }else wait;
     } else wait;
  }
  msgsrv getAck(int srcPort){
     buffer[srcPort]--;
     bufferFull[srcPort]= false;
     outEnable[sender] = true;
     inEnable[srcPort] = true;
  }
  msgsrv giveAck(int dstX, int dstY, int srcPort, int packId){
     int port = sender.portNumber;
     if(dstX==Xid && dstY ==Yid) { //packet reached its destination
        if(packId==predefinedNum )
           packetReceived = true;
        sender.getAck(srcPort) after(t4);
     }else if(! bufferFull[port]){
        sender. getAck(srcPort) after(t4);
        self.reqSend(dstX,dstY,srcPort,packId);
     }else wait;
   }
}
main(){
  Router r00(r03,r10,r01,r30):(0,0);
}
```

Figure 1: Pseudo code for Rebeca model of ASPIN

the timing behaviors. Considering these details, the full state space for a specific traffic pattern will be produced and checked for deadlock and the successful arrival of each packet. In order to estimate maximum end-to-end packet latency, buffer delays and channel delays are also considered in the model. Using an actor-based modeling language we can efficiently map the constituents of GALS NoC, to actor model. Different elements of a GALS NoC can be modeled as follows,

- **Router**: Each router can be mapped to an actor which communicates with other routers through message passing. Delay for processing in a router, e. g. executing routing or scheduling algorithms can be modeled by ”delay” construct. We can define some message servers to model routing algorithms. An actor in Rebeca model is able to recognize who has invoked its message server, thus the router can understand from which port a packet entered to a router and then decide to which router the packet should be sent. Similarly, to model scheduling algorithms we can use message servers. Since ASPIN uses Round Robin scheduling algorithm, here we simply use the scheduler of Timed Rebeca.
- **Buffer**: Router buffers can be seen as an array of elements (packets). We can use Rebec queues to model buffers, and then keep track of the number of packets in the buffer by defining a state variable as a counter for the number of elements in the buffer. Doing so, we always have the number of packets in the buffer, thus being able to model adaptive and dynamic routings.

Delay of writing and reading to/from buffers can be modeled by ”after” constructs.

- **Packet**: Since our work concentrate on analysis of the interconnection we only model a packet with its identifier and its destination. We can use ”deadline” construct of Timed Rebeca, if a packet has a specific deadline.
- **Channel (Link)**: Channels can be simply modeled by message passing. Delay of passing through a channel can be modeled using ”after” construct.
- **Communication protocol**: by defining appropriate message servers, we can model communication protocols of a GALS NoC.

Figure 1 shows a pseudo code for the Rebeca model of ASPIN. The code is available in [NoC13]. As shown in the pseudo code the model consists of one reactive class Router, and a main part. Instantiated rebecs of the reactive class Router are declared in the main part (as shown for router *r00* in Figure 1). The list of known rebecs and the parameters for initializing a rebec (here, the address of a router) are given in the instantiation statement. In the pseudo code we only show instantiation of one router, more routers can be defined like *r00* to model a larger NoC. Packets are generated in the Router method (the initial method) of each router. Packets can be generated at any time. As illustrated in the pseudo code one packet is generated in *r00* at time t_1 and another is generated in *r20* at time t_2. Each packet only has a destination address and identifier. Packets are transferred through channels, using four-phase handshake communication protocol. We model channel functionalities by means of message passing in Rebeca. Four-phase handshake protocol is modeled using three message servers: *reqSend*, *giveAck*, and *getAck*. A router calls its *reqSend* message server to send a request to its neighbors; *reqSend* requires as parameter, a direction (*srcPort*) that determines in which input buffer the packet is stored, a destination address (*dstX* and *dstY*) that shows the destination of the packet and the packet Identifier (*packId*). The function *XY-routing* selects which neighbor router the packet should be sent to, and then *giveAck* message server of the selected neighbor router is called.

Following four-phase protocol, request signal of the sender is raised until it receives an acknowledgment signal. While waiting for the acknowledgment signal, the router cannot send any packet from that port. We assigned *outEnable* Boolean to each output port of the router to model this functionality. Whenever a packet is sent from an output port, the corresponding *outEnable* becomes false. For sending the acknowledgment signal the *getAck* message server is invoked. The *giveAck* message server first checks the address of the destination of the packet. If the address is the same as the current router, then the rebec will consume the packet. Otherwise, it checks if the corresponding input buffer has enough capacity to store the packet, if there is enough capacity the packet will be stored and an acknowledgement is sent to the sender by calling its *getAck* message server. Then, it will be sent to the other neighboring routers calling their *reqSend* message server. If the buffer is full the packet will not be stored in receiver buffer and the sender should wait for an acknowledgement from the receiver until the buffer has an empty space.

To model the behavior of router buffers, we use the rebec queue to store all packets received by a router and only keep track of the length of North, South, East and West buffers to have buffer status at all time. Two arrays *buffer* and *bufferFull* are used to store the length of each buffer and their status. In two message servers, *reqSend* and *giveAck*, the length of buffers are changed. Writing and reading delays are also considered for buffers. Writing delay points to the delay between entrance of a packet to an input buffer and the moment that a request is sent for an output port, i.e. it includes routing delay of the packet. Reading delay refers to the time between a request to an output port and sending the packet to the neighbor router. In the pseudo code, t_3 is the sum of writing delay, link delay and reading delay and t_4 is link delay. To model storage strategy of input buffering we assign *inputEnable* Boolean variable to each input buffer to prevent sending a packet from an input port before getting the acknowledgement of the previously sent packet.

5 Functional and Performance Analysis

In this phase, the model checker generates the whole state space, and then traces all execution paths of the system to check if the given property is satisfied. Each execution path shows a possible order of packet movement in the system. Thus, a model checker checks all possible orders of packet movement in contrast to simulation techniques that checks random execution paths, and hence random orders of packet movement.

We use Afra [Reb], the model checker of Rebeca, for verification of the specified functional and performance properties. Verification of the properties using model checking is discussed in Subsection 5.1 and Subsection 5.2. To tackle the problem of increasing number of states in large NoCs, a method based on compositional verification is presented in Subsection 5.3.

5.1 Functional Verification

Deadlock Freedom: Verifying the model against deadlock freedom is straightforward in Afra. Afra traces all possible states of the model to check for deadlock. A model has deadlock if it contains a state that has no outgoing transition. If a deadlock occurs in a NoC it means that some packets can no more be transmitted forward.

Successful Packet Sending: We consider two correctness properties regarding successful packet sending. The first property, called Correctness 1 is to check whether a packet sent from a source has reached its destination. For checking this property two state variables of type Boolean, namely *packetSent* and *packetReceived* are added to the router model (Figure 1). In each router, if a predefined packet is sent then *packetSent* becomes true, and if that packet is received by its destination then *packetRecieved* of the destination becomes true. Also, it is needed to check whether all packets in the network were sent through counting the number of sent packets. To do so, a Boolean state variable *allSent* should be declared. It will become ”true” if the number of sent packets is equal to the number of existing packets in the NoC. This property is referred to as Correctness 2. To count the number of sent packets we can use a reactive class that is known by all routers in the system and inform it each time a packet is sent. Therefore, to ensure the successful arrival of all packets, the model checker traces all execution paths to check if the following two properties holds in all of them,

Correctness 1 : ”If *packetSent* state variable of the source router of a packet becomes true, then *packetReceived* state variable of the destination router will eventually become true.” (It can be specified in LTL as: $G\ (packetSent \rightarrow F(packetReceived)\ )$.)

Correctness 2 : ”Eventually *allSent* state variable becomes true.” (Specification in LTL: $F(allSent)$.)

5.2 Performance Evaluation

Estimation of the Maximum End-to-End Latency: Many packets may be produced in cores and transferred to other cores. They may make disruption for each other when multiple packets are competing to gain an output port at the same time; thus a packet may reach its destination with different latencies. To gain the maximum end-to-end latency of a packet, we can add a Boolean variable that becomes true when the packet reaches its destination, and then check the variable after some time. We defined *checkReceived* message server and call it with an *after(t)* command, just after sending the packet. Thus the model checker checks if in all execution paths the targeted packet is received after t times. In Section 6, some scenarios are introduced and verified to show how our method considers delays made by disruptions to a target packet. We also have an example to show how our method can be used in making design decisions. We evaluate the best memory location in an 8×8 ASPIN model such that we have the least maximum memory access time.

5.3 Compositional Method to Estimate Maximum End-to-End Latency

The idea behind this method is to divide the path into sub-paths and then estimate the maximum latency by evaluating latency of packets in each sub-path. To do so, possible packet paths should be investigated precisely to find out which packets may make disruption for a transferring packet. According to the XY routing algorithm a packet path is composed of two main sub-paths; vertical sub-path and horizontal sub-path. In the following, these sub-paths are investigated to know how the latency of a packet moving in these sub-paths depends on other moving packets.

Horizontal Sub-path. When a packet P moves along a row, it may be disrupted by two kinds of disrupting packets as follows:

(a) *Dependency 1*, Packets moving in the same row: They may make disruption if they want to get to the same port as P. For example if a disrupting packet arrives at a port just before P, it will occupy the port and P should wait until the disrupting packet leaves the port, because the size of output buffer is equal to 1. On the other hand because of using input buffering as storage strategy, P may be delayed by packets that arrive in the input buffer before P, until they are sent and their acknowledgements are received. So, in a row both packets moving in the same direction as P and in the reverse direction may make disruption to packet P if they try to gain the South or North ports.

(b) *Dependency 2*, Packets moving in other rows: A disrupting packet in a row may itself be delayed by other packets; thus if a disrupting packet wants to turn to a column it may be delayed by other packets in that column which are originated from other rows. This way, the latency of P may depend on packets of other rows.

Vertical Sub-path. If a packet P moves in a column, it may be disrupted by packets as below:

(a). *Dependency 3*, Packets moving in the same column. According to XY routing algorithm, packets of different rows can enter a column and compete to acquire ports. Only packets that move in the same direction as P are able to make disruption to packet P, because, there are two channels with reverse directions connecting two neighbor routers to each other.

(b) *Dependency 4*, Packets moving in other columns. As discussed before, packets of different rows can enter a column. Because different columns may have impact on the latency of the packets of a row, latency of packet P while moving in a particular column relates to other columns.

The compositional method should consider all the specified dependencies to work correctly. Knowing the generation time of all packets, the compositional method gives an upper bound for the maximum end-to-end latency of a packet P, initiated from (r_1, c_1) to reach its destination in (r_2, c_2), where r_1, c_1 and r_2, c_2 are row and column numbers of source and destination of the packet, respectively. The method is composed of two steps as follows:

Step 1 (Decomposition step). In this step all rows of a $m \times n$ NoC should be modeled as a $1 \times n$ NoC (row models only differs from each other in the location and the time for generation of packets) and then verified separately to determine when packets with their destination placed in column c_2 will arrive at column c_2. For verification of a row, all possible orders of packets to gain a port are considered; thus, disruptions of packets moving in the same row as P *(*r_1*)* are considered (Dependency 1). As discussed before, the end-to-end latency of packet P may be influenced by packets of other rows, if P is stored in an input buffer waiting for other packets in the same input buffer wanting to turn to a column. We assume that in this situation P waits at most t_1 time units were t_1 is the maximum time in the worst case with respect to the number of packets moving in that column. In the worst case P should wait for all packets in that column to be able to move to the next router in its path. By this approximation we consider the dependency of packets moving in other rows (Dependency 2). The approximation can be more precise by respecting the time that packets start moving along the column, but here we only introduce the method using a simple and upper bound approximation.

Step 2 (Composition step). In this step column c_2 is verified by taking into consideration all packets moving in the same direction as P in this column; hence, all disruptions that may be caused by packets in c_2 are respected (Dependency 3). As stated in step 1, we assume that P waits t_1 time units if it is disrupted with a packet, which wants to turn to a column. In fact doing this work we may uncover some dependencies because we have only considered the maximum delay t_1 in r_1 , however the same situation may occur for other packets in other rows and they would make disruption for packet P, thus we may not consider all possible disruptions if we consider delay t_1 only in r_1. To cover these situations we also consider the case where other packets in *(r, c)* are delayed when they are in the same situation, by some t time units. Where $t = t_1$ - d , and d is the time needed for a packet to move from r_1 to r strictly (Dependency 4). For example if r_1 - r = dr and passing from a router to its next router takes x times, $d = dr \times x$. The case is shown in Figure 2, the congestion caused by packet (6) and (7) results in delay in packet (5) and thus packet (5) makes disruption for packet (1).

6 Results

Results for functional verification and estimating the maximum latency are presented in Subsection 6.1 and Subsection 6.2. In Subsection 6.3 evaluation of memory location in NoC 8×8 using our method is explained. In the following, five scenarios are introduced for a 4×4 ASPIN. The first two scenarios are very similar. We only perform functional verification on them to show a property violation in the system. The last three are verified against functional properties and

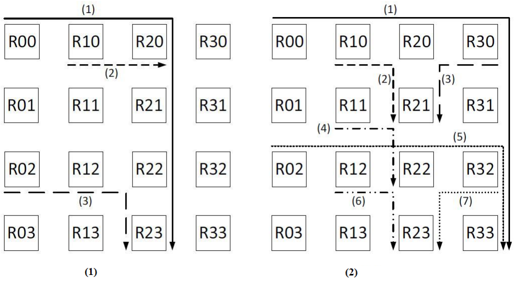


Figure 2: 1) Scenario 4, 2) Scenario 5

Scenario	*Deadlock freedom*	*Correctness1*	*Correctness2*
1	12	14	14
2	Not satisfied	Not satisfied	Not satisfied
3	24	25	25
4	193	194	194
5	2240	2241	2469

Table 1: Results (number of states) for functional verification

also the maximum end-to-end latency for a target packet, namely packet (1), is estimated. These scenarios are selected in a way that covers all the dependencies introduced in Subsection 5.3.

Scenario 1: Three packets are generated; two packets sent from routers R00 and one from R10 to R11.
Scenario 2: In this scenario we assume that due to some faults in scenario 1, R10 does not send an acknowledgement.
Scenario 3: Packet (1) is sent from R00 to R23, and no other packets will make disruption.
Scenario 4: This scenario shows how the packet can be delayed in a row and in a column. Router R10 generates packet (2) as soon as it receives packet (1), thus packet (2) may cause disruption to packet (1). On the other hand R02 produces packet (3) in a way it reaches R22 at the same time as packet (1), so packet (1) may be delayed by packet (3) too (Figure 2).
Scenario 5: Packet (1) is disrupted by packet (2), and packet (2) is itself disrupted because of congestion in R21 (impact of congestion in a row on other rows). On the other hand, congestion in R23 leads packet (5) to be blocked until packet (4) leaves the input port of R22. This may results in disruption of packet (1) by packet (5), if they reach R32 at the same time (impact of congestion in a column on other columns). (Figure 2)

6.1 Results for Functional Verification

Results for verifying the model against Deadlock freedom and Successful arrival of packets are shown in Table 1. In each scenario, the model checker traces the full state space. The model checking time for each scenario is between 1 to 3 seconds. As illustrated in Table 1 scenario 2

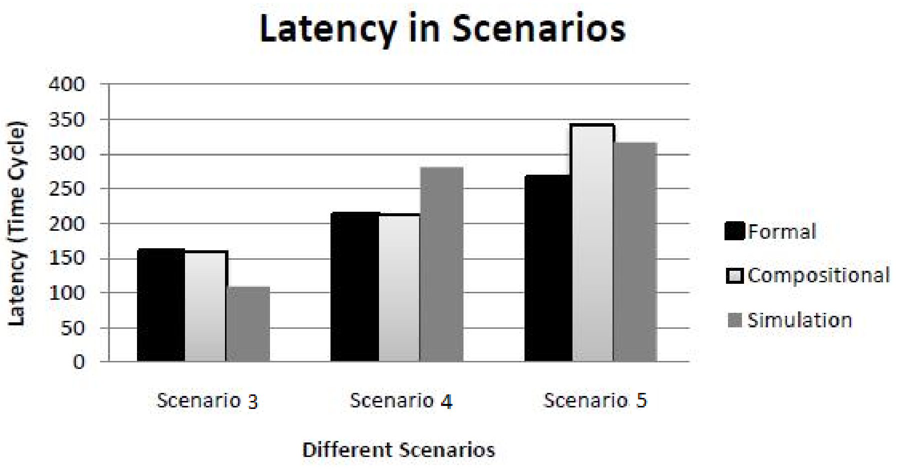


Figure 3: Comparison of results for estimating latency using simulation, model checking and compositional methods for scenarios 3 to 5

contains deadlock, because some packets wait forever. Therefore Correctness 2 property is also violated and Correctness 1 only is satisfied for the packet sent from R10. All other scenarios satisfy specified functional properties.

6.2 Results for Estimating the Maximum Latency

We used HSPICE[1] simulation to validate our formal and compositional methods. As shown in this section, the results of simulation match the results from formal and compositional methods while effort for simulation is much higher. The reason is of course the more details that are considered in HSPICE. The similarity of the results shows that in spite of the fact that our methods are not considering the same amount of details they are still eligible for the required measurements. By this comparison we show how using our method in the early stages of design can help the designer to make suitable architectural decisions according to the performance parameters of the system.

To compare our results to simulation results, we extract buffer read and buffer write delays from HSPICE simulation of ASPIN for 32nm CMOS technology and normalized to C-element Muller gate to convert them from Float to Integer. Delay of read and write operations on a buffer are considered 19 and 6 time units, respectively. Producing and consuming a packet in a core cause delays of 10 and 19 time units, respectively. We consider capacity of 2 entries for input buffers and 1 for output buffers. Figure 3 shows results for estimating the maximum latency of packet (1) in the specified scenarios compared to results of the simulation and results of estimation using our compositional method. Both formal and compositional approaches consider all execution paths of the scenarios to find the maximum latency for packet (1). Using Afra tool we can find the execution path that violates a specified property. We use this capability and

[1] The lowest level of simulation in hardware domain are done in HSPICE simulator. In this tool all details of transistors and wires are considered.

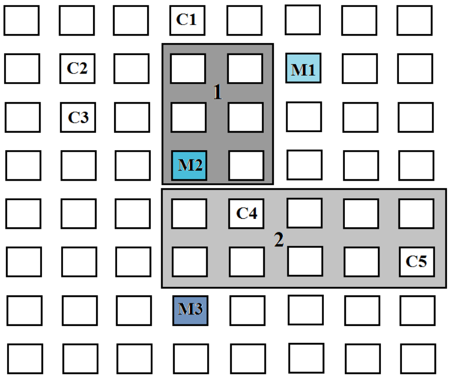


Figure 4: Example of memory locations in an 8 × 8 ASPIN; Cores C1 to C5 are the cores in which memory access is a concern; M1, M2 and M3 are the candidate cores for memory placement; Areas 1 and 2 are highly congested.

extract the execution path which generates the maximum end-to-end latency. To do so, we can first use the method presented in Subsection 5.2 to find the maximum latency, say L_{max}, and then check the following property: ”The packet arrives at destination in less than L_{max} time units.” Obviously, the path with the maximum latency violates the above property and can be extracted. Then we execute HSPICE simulator for the specified execution path to get the result for maximum end-to-end latency for the same packet. Running each scenario in HSPICE that traces only one path takes more than 24 hours on a system with Corei7, 2.6 GHz processor and 24GB of memory. In contrast running each scenario using formal techniques takes about 1 to 3 seconds. For the compositional method, it takes 1 to 3 seconds for each row and one column.

Results show packet latency increases when disruption occurs, but the amount of increase in latency is not the same in all three methods (simulation, model checking and the compositional method). The reason is that some delays in the system like the delay of scheduling have not been considered in the formal model. In addition delay of routing varies in different scenarios according to the number of gates being used in the router. Due to considering a fix routing delay in our formal model, timing results for formal verification of scenario 3 is greater than that of simulation (routing delay in some routers are less than our assumption). Comparison between the results of formal method and our compositional method shows that the compositional method can estimate an upper bound for maximum latency.

6.3 Evaluation of memory location in NoC

We use 8 × 8 ASPIN model to perform three experiments to evaluate the best memory location, among three choices. Lets assume that we have five cores for which memory access time is a concern due to their timing requirements. The goal is to choose the memory location that results in the least memory access time for these cores, under specific traffic pattern of an application. For each experiment, we use a different memory location, and thus the traffic pattern is the same except for memory requests. Packets are almost evenly generated in the network; however the congestion rate in areas 1 and 2, shown in Figure 4, is higher than other areas. To perform the

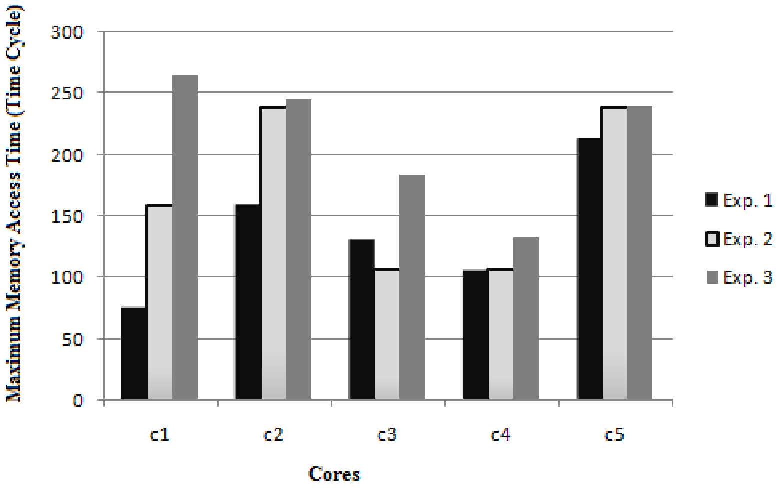


Figure 5: Experimental results showing the maximum access times for each core, c1 to c5, in the three experiments with different memory locations.

experiments we inject 40 packets in the network and set the packet generation time intervals in the source cores in a way that results in high congestion in the areas 1 and 2. In the experiments Exp. 1, Exp. 2, and Exp. 3 the locations M1, M2 and M3 are assumed to be where the memories are located respectively. Figure 4 illustrates the configuration of memory locations and the cores. Under the specified traffic pattern the model checker verifies all possible order of packet movement to find the maximum memory access time for each of the cores. Number of states and transitions grow proportionally with the congestion in the system.

Memory placement will be affected by the routing algorithm and also traffic pattern used by an application. Figure 5 shows that choosing M1 as memory location results in minimum average maximum access times. Although M2 has the least average distance from the cores, the fact that it is placed in the congestion area 1, and that we use XY routing algorithm causes the average maximum latencies to be increased in Exp. 2 compared to Exp. 1. In fact using deterministic routing algorithm XY leads to increase latency in highly congested traffic patterns. Our method helps designer make suitable design decisions (e.g. find the best memory locations) according to other design parameters, and performance requirements.

7 Conclusions

We presented a formal model based on actor model for a GALS NoC that can be analyzed from two aspects of functionality and performance. The model was verified against two functional properties, and also for estimating the maximum end-to-end packet latency. To alleviate the problem of state explosion in the case of large NoCs, we presented a method based on compositional verification specific for NoCs that uses XY routing, which can be applied to estimate an upper bound for the maximum end-to-end packet latency. In an example we showed how our method can be used in making design decisions for memory location based on performance evaluation. Using formal methods enable us to apply an exhaustive verification in the early stages of design in contrast to methods based on simulation. Comparing to other works based on formal and mathematical techniques, our model considers hardware concept like buffer and link delays. Therefore, being able to perform buffer analysis we can model adaptive and dynamic routings. We used an approximation in our compositional method to predict the impact of a column on the latency of a packet moving along a row. As a future work the approximation can be more accurate by considering the situation of packets of that column precisely. More work on performance evaluation for answering more variety of questions is one of the main future directions of this work.

Bibliography

[BHHK10] C. Baier, B. R. Haverkort, H. Hermanns, J.-P. Katoen. Performance evaluation and model checking join forces. *Communications of the ACM Journal* 53(9):76–85, sept 2010.

[Bul10] Validation of Multiprocessor Multithreaded Architectures (MULTIVAL). 2010. Bull, CEA/LETI, INRIA and ST Microelectronics.
http://vasy.inria.fr/multival/

[CHLS10] N. Coste, H. Hermanns, E. Lantreibecq, W. Serwe. Towards performance prediction of compositional models in GALS designs. In *Proc. CAV Symposium on the Theory of Computing*. Pp. 204–218. 2010.

[CSH$^+$10] Y. Chenl, W. SU, P. Hsiungt, Y. Lan, Y. Hu, S. Chen. Formal modeling and verification for network-on-chip. In *ICGCS Symposium on the Theory of Computing*. Pp. 299–304. 2010.

[Das08] S. Dasgupta. *Formal Design and Synthesis of GALS Architectures*. PhD thesis, Newcastle University, 2008.

[FTHJ09] S. Foroutan, Y. Thonnart, R. Hersemeule, A. Jerraya. Analytical computation of packet latency in 2D-Mesh NoC. In *proc. TAISA Symposium on the Theory of Computing*. Pp. 1–4. 2009.

[GAE11] Globally Asynchronous Elastic Logic Synthesis (GAELS): Case for Support. 2011. University of Manchesterand University of Newcastle, Project.
apt.cs.man.ac.uk/case_for_support-final.pdf

[Gem11] The gem5 simulator. 2011.
dx.doi.org/10.1145/2024716.2024718

[Hew72] C. Hewitt. *Description and Theoretical Analysis (Using Schemata) of PLANNER: A Language forProving Theorems and Manipulating Models in a Robot*. MIT Artificial Intelligence (USA), 1972.

[ITR11] International Technology Roadmap for Semiconductors- ITRS. 2011.
http://www.manmaker.com/manual/

[KKK$^+$12] E. Khamespanah, Z. S. Kaviani, R. Khosravi, M. Sirjani, M. Izadi. TimedRebeca Schedulability and Deadlock-Freedom Analysis Using Floating-Time Transition System. In *Proc. AGERE! Symposium on the Theory of Computing*. Pp. 23–34. 2012.

[KPSD11] M. Kamali, L. Petre, K. Sere, M. Daneshtalab. Refinement-based modeling of 3D NoCs. In *Proc. FSEN Symposium on the Theory of Computing*. Pp. 236–252. 2011.

[Mad09] G. Madl. *Model-based Analysis of Event-driven Distributed Real-time Embedded Systems*. PhD thesis, University of California, 2009.

[MS10] R. Mateescu, W. Serwe. A study of shared-memory mutual exclusion protocols using CADP. In *Proc. FMICS Symposium on the Theory of Computing*. Pp. 180–197. 2010.

[NB12] M. Najibi, P. Beerel. Performance bounds of asynchronous circuits with mode-based conditional behavior. In *ASYNC Symposium on the Theory of Computing*. Pp. 9–16. 2012.

[NIR] NIRGAM.
http://nirgam.ecs.soton.ac.uk/

[NoC13] Rebeca model for NoC 4x4. feb 2013.
http://www.rebeca-lang.org/wiki/pmwiki.php/Examples/NOC4x4

[Reb] Rebeca Formal Modeling Language.
http://www.rebeca-lang.org

[SAB04] M. Sirjani, A. M. nad A. Shali, F. de Boer. Modeling and Verification of Reactive Systems using Rebeca. *Fundamenta Informaticae Journal* 63(4):385–410, Jun 2004.

[SF01] J. Sparso, S. Furber. *Principles of Asynchronous Circuit Design A System Perspective*. Kluwer Academic Publishers (London), 2001.

[SGM08] A. Sheibanirad, A. Greiner, I. Miro-Panades. Multisynchronous and fully asynchronous NoCs for GALS architectures. In *IEEE. Design Test of Computers Symposium on the Theory of Computing*. Pp. 572–580. 2008.

[SJ11] M. Sirjani, M. M. Jaghoori. Ten Years of Analyzing Actors: Rebeca Experience. In *Formal Modeling Symposium on the Theory of Computing*. Pp. 20–56. 2011.