Comparison of NoC Routing Algorithms Using Formal Methods

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

(z.sharifi@ut.ac.ir, smohammadi@ece.ut.ac.ir, marjan@ru.is)

[1] School of Electrical and Computer Engineering, University of Tehran, North Karegar, Pardis 2, Tehran, Iran

[2] School of Computer Science, Reykjavik University, Menntavegi 1, 101 Reykjavik, Iceland,

[*] Contact author

This paper is submitted to PDPTA'13

Abstract: Network on Chip (NoC) has emerged as a promising interconnection paradigm for complex on-chip communications. As fabrication cost is high, model based design of NoC and early exploration to make proper design decisions are important challenges in NoCs. To tackle these challenges, we use formal methods and utilize their expressivity and flexibility to model different behaviors of a NoC and their abstraction to support early analysis of the design. We propose a formal approach for selection of the best routing algorithm in a NoC, according to its performance requirements. We present a model for two-dimensional mesh NoC using actor based modeling language Rebeca. Both functional and timing behaviors are modeled. The model is then used to compare three routing algorithms XY, Odd-Even and DyAD with respect to the maximum end-to-end packet latency in different scenarios.

Keywords: Network on Chip (NoC), model checking, Rebeca, routing algorithm, performance evaluation

1. Introduction

Asynchronous paradigm has become conspicuous in Network on Chip (NoC) design to overcome problems of clock skew and clock tree distribution of fully synchronous design. Thereby Globally Asynchronous Locally Synchronous (GALS) NoC has gained attention in design of such systems [1]. Functional verification is a major challenge in these systems to avoid increase in design errors; but a functionally verified GALS NoC may not meet all its desired performance. Thus, performance prediction in the various stages of the design is another necessity that should be performed to help the designer make proper design decisions according to the parameters of the system and also performance requirements. One important design decision for systems where end-to-end latency is a concern is to select a routing algorithm that results in the least end-to-end latency.

As fabrication cost is high, it is desirable to perform analysis on NoC design before having the first prototype and even in the early stages of design process. For model-based analysis we need to capture the crucial details in the model. However, to the best of our knowledge existing models of GALS NoC do not present the required details for modeling adaptive, dynamic and deterministic routings.

One important point in asynchronous systems is that lack of a reference clock leads in 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, 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.

In this paper, we use model checking for performance prediction on two-dimensional mesh GALS NoC[1]. Model checking is a promising approach that can be used for both performance evaluation and correctness checking and allow us to perform exhaustive search in the state space [2]. Important advantages of using model checking for performance prediction, in the case of this work are:

- Expressiveness: by using a suitable modeling language we can simply model both functional and timing behavior of GALS NoC, and also consider asynchronous paradigm and nondeterministic behavior of the system.
- Abstraction: for higher efficiency and for verifying more complicated properties we can model only the necessary details with respect to the property and abstract away the irrelevant parts. Abstraction enables us to perform analysis in the various stages of the design flow.
- Exhaustive verification: given the model of the system and the targeted properties, model checker

[1] We have shown in a paper in submission that the model can be used for functional verification at the same time.

explores the whole state space to check for property satisfaction rather than a set of traces.

- Finding the Finding the violating execution path: Model checker can return the execution path in which the property is violated (in contrast to mathematical and analytical approaches), and thus can help the designer for improving the design.

We used Timed Rebeca (Reactive Objects Language) [3, 4, 5] 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. Consistency between the computational model of Rebeca and GALS NoC, enables us to model a GALS NoC naturally and simply. Each router in a GALS NoC is modeled as an actor and the communication between routers are modeled as message passing between actors.

To estimate the maximum end-to-end packet latency, the delay for read/write from/to a buffer, and delays of links and routing are considered in the model. Four-phase handshake communication protocol is also modeled for communication through channels. To model different kinds of routing algorithms, especially adaptive and dynamic algorithms, we capture buffer statuses (number of elements in the buffers). Subsequently, the model is used for comparison of some routing algorithms, namely XY, Odd-Even and DyAD. Results of comparison can be further used by designers to take proper decision about routing algorithms in the early phases of design.

The remainder of the paper is organized as follows. In Section 2 related work is introduced, Section 3 contains preliminaries. Section 4 presents GALS NoC model in Rebeca. Three routing algorithms are introduced and modeled in Section 5. Results are shown in Section 6, and finally conclusion and future work are presented.

2. Related Work

There exist many simulation based works on analysis of different aspects of NoCs. Various arbitration and routing algorithms, router switches, and traffic patterns have been modeled using simulators. Nirgam [6] and gem5 [7] are two simulators for analyzing NoCs. In [8] a simulation based method for deadlock detection in a multiprocessor system with many running processes is proposed. As discussed before, simulation based methods are non-exhaustive and cannot be applied for early exploration because they do not have the adequate level of abstraction.

Formal and mathematical approaches are able to perform exhaustive verification at the expense of losing some precision. There are some works based on mathematical approaches; such as [9], which uses deductive method to prove that a routing algorithm is deadlock free. Although mathematical techniques are powerful, they cannot show how a violation occurred in the system. Formal methods are able to address this challenge.

There exist formal tools used for functional verification and performance prediction of the same model simultaneously. Formal techniques have been widely used for analysis of different aspects of multiprocessor systems that are in close relation with NoCs. A Petri net model is presented in [10] for performance modeling of asynchronous circuits. In [11] and [12] multiprocessor systems have been modeled by Timed Automata considering bus based methodology as interconnect network. In none of the above works GALS NoC was analyzed; GALS NoC has many special timing details and complex modules.

In [13] a NoC is modeled in Extended Timed Automata, and its router is verified against some functional properties. Authors in [14] applied Interactive Markov Chain (IMC) and Interactive Probabilistic Chain (IPC) to model a buffer used in NoC design. However, details of hardware timing and link model are not mentioned. In [15] 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. Using probabilities reduces the state space at the expense of losing the buffer analysis.

In this paper, we use formal methods to model different kinds of routing algorithms. The comparison is performed with respect to the maximum end-to-end packet latencies. In contrast to existing works based on formal methods, our model considers hardware details like link and buffer (read and write) delays and buffer statuses and thus can model adaptive and dynamic routing algorithms. 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

Here, Timed Rebeca is introduced as the modeling language used for our analysis.

3.1 Timed Rebeca

Timed Rebeca is an extension to Rebeca, capable of modeling functional and timing behaviors of distributed reactive systems.

Rebeca is an actor based modeling language [16] 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. Communication is asynchronously established through message passing. Message passing is fair and implemented by method calls; calling a method of a rebec results in sending a message to the actor that invokes corresponding message server. Each rebec has a buffer, called a queue, for arriving messages. In each step a rebec is executed by

```
Model ::= Class* Main
Main ::= main { InstanceDcl* }
InstanceDcl ::= className rebecName(⟨rebecName⟩*) : (⟨literal⟩*);
Class ::= reactiveclass className { KnownRebecs Vars MsgSrv* }
KnownRebecs ::= knownrebecs { VarDcl* }
Vars ::= statevars { VarDcl* }
VarDcl ::= type ⟨v⟩+;
MsgSrv ::= msgsrv methodName(⟨type v⟩*) { Stmt* }
Stmt ::= v = e; | Call; | if (e) { Stmt* } [else { Stmt* }] | delay(t);
Call ::= rebecName.methodName(⟨e⟩*) [after(t)] [deadline(t)]
```

Fig. 1: Syntax for Timed Rebeca

removing a message from the top of its queue and executing its corresponding message server.

To model timing behaviors of a system, three constructs are provided as follows:

- *delay (t)*: causes a delay of *t* time units.
- *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.
- *deadline (t)*: this construct is paired with a method call, and the corresponding message will be deleted from the queue after t time units.

Abstract syntax of Timed Rebeca is illustrated in Fig. 1.

4. GALS NoC Model

If the model is too abstract, results may become imprecise; on the other hand very low level of abstraction may intensively increase complexity and leads analysis to state explosion. Using the proper abstraction level is the key for model based analysis of NoC. To this end one should define the constituent of the model with respect to the properties that the model is verified against.

In this paper we target maximum end-to-end packet latency for comparison of different routing algorithms. Network topology, router buffers, routing algorithm, communication policy, storage strategy and channels are modeled. Timing behaviors like link delay and the delay for writing and reading to/from buffers 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 modeled as an actor which communicates with other routers through message passing. Delay for processing in a router, e. g. scheduling or routing algorithms can be modeled by "*delay*" construct.
- Routing algorithm: 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 and then decide to which router the packet should be sent.

```
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);
}
```

Fig. 2: Pseudo code for GALS NoC model

- 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: we model a packet only with its identifier and its destination.

- 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.

Fig. 2 shows a pseudo code for our model for GALS NoC 4×4 in Rebeca. The code is not limited to 4×4 NoCs and can be used for larger ones provided that we take into consideration the problem of state space explosion. The code is available in [17]. According to the pseudo code the model consists of one reactive class Router, and sixteen instantiated rebecs namely r00, r01, r02 to r33.

Packets are generated in initial message server of routers. Each packet only contains its destination address and no data are modeled, because only analysis of communication part of a NoC is of interest. Packets transfer through channels, using four-phase handshake communication protocol. We modeled channel functionalities by means of message passing capability of 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 that determines in which input buffer the packet is stored and a destination address that shows the destination of the packet. Routing algorithm selects which neighbor router the packet should be sent to, and then *giveAck* message server of the selected neighbor router is called. *giveAck* first checks if the corresponding input buffer have enough capacity to store the packet, if it does, the packet will be stored and an acknowledgement is sent to the sender by calling its *getAck* message server. Then, it will be either consumed or sent to other neighboring routers using *reqSend* message server. While the buffer is full the packet will not be stored and should wait until the buffer has an empty place.

In two *reqSend* and *giveAck* message servers the length of the buffer can change. when a packet is inserted or deleted from the buffer. Writing and reading delays are also considered for buffers.

5. Model for Routing Algorithm

Routing algorithms can be classified into *deterministic* and *adaptive* routings. In a deterministic routing there can only be one path between a source and a destination, whereas in adaptive routing more than one possible path may exist and the algorithm considers dynamic network condition to decide in which direction a packet should be transferred.

In the following sub-sections we present a formal model for XY and Odd-Even routing algorithms as instances for deterministic and adaptive routings respectively. Dynamic Adaptive Deterministic (DyAD) is also modeled.

5.1 XY Routing

In this algorithm, first packets move along X direction to get to the column of the destination, and then along Y direction to reach their destination.

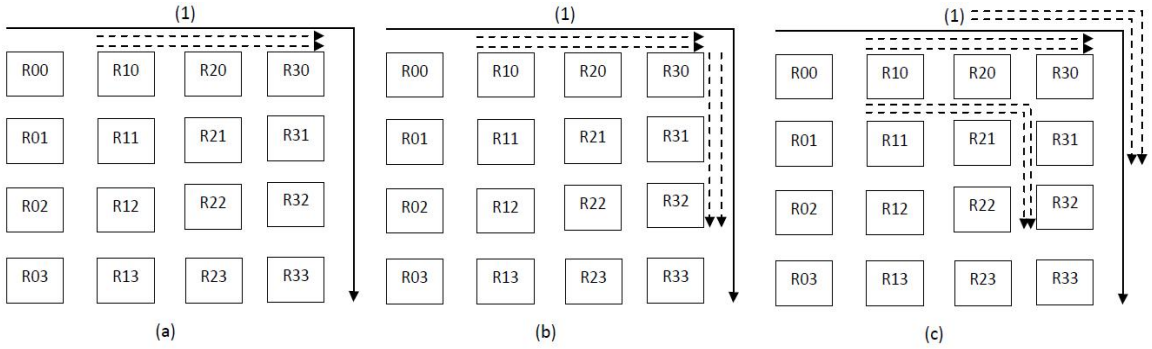

Fig. 3: Scenarios 1, 2 and 3.

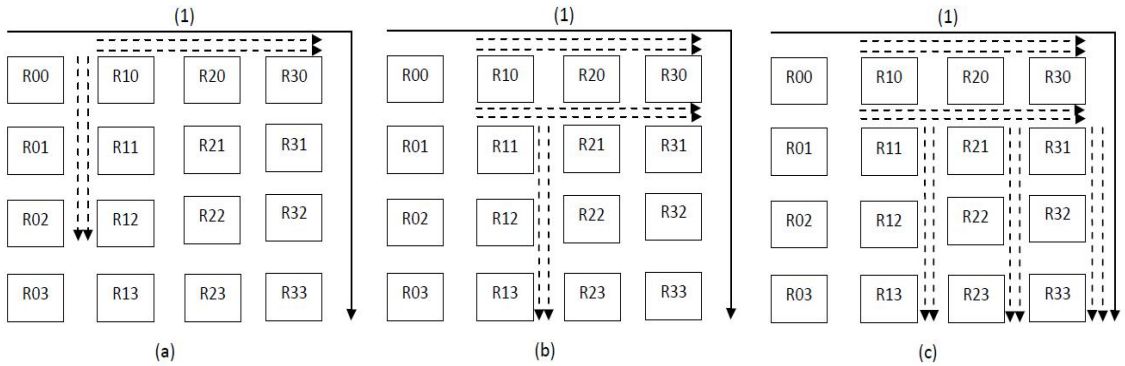

Fig. 4: Scenarios 4, 5 and 6.

To model this algorithm, a router (X,Y) compares its X location to that of the packet destination, if it is

greater/smaller, it calls the *giveAck* method of west/east neighbor. The same approach is done for the Y coordinate.

5.2 Odd-Even Routing

Odd-Even routing is an adaptive routing algorithm based on Odd-Even turn model [18]. Odd-Even turn model restricts the turns in the packet path to ensure about the deadlock freedom. According to Odd-Even turn model *north-to-west* and *south-to-west* turns are prohibited in routers located in an odd column and *east-to-south* and *east-to-north* turns are prohibited in routers located in an even column.

Among possible directions where an Odd-Even router can send packet, the direction in which the downstream router has less empty slots in its corresponding input buffer is selected.

In this algorithm each router keeps track of the number of packets in input buffer of each of its neighbors. In our model whenever the size of an input buffer of a router changes, it informs its corresponding upstream neighbor by sending a message.

5.3 DyAD Routing

DyAD routing dynamically uses a deterministic or an adaptive routing exploiting both of them in different network congestion conditions.

Each router monitors the occupation ratio of its input buffers (except for the local buffer). Whenever one of the buffers reaches a predefined *congestion threshold* a *mode* flag is set to inform the corresponding neighboring router about the congestion. On the other hand, each router continuously checks mode flag of its neighbors to decide whether to work with deterministic or adaptive routing. According to [19] if at least one of the neighboring routers were congested the router would decide to work with adaptive routing; otherwise it would work with deterministic routing.

To model a DyAD router we add a mode flag to our model. The mode flag becomes true if the size of the corresponding input buffer reaches the congestion threshold.

6. Results

We use Afra [20] tool for model checking of XY, Odd-Even and DyAD Rebeca models, and compare them with respect to the maximum latency of the target packet. The NoC size in these comparisons is 4×4. All input buffers are of size 3 packets and %33 congestion threshold.

To compare the three algorithms we introduce six different scenarios describing different network conditions. In all scenarios the target packet is packet (1). We call the path of a packet to its destination *R-path*, when it is routed by the routing algorithm *R*.

The scenarios are as follows:

Scenario 1. Router R10 generates two packets[1] as soon as it receives packet (1). The two packets may cause disruption for packet (1) (Fig. 3. a).

Scenario 2. Each of the routers R10 and R30 sends two packets to R30 and R33. This may cause disruption to any packet transferring from their paths (Fig. 3. b).

Scenario 3. Three routers R10, R20 and R21 send packets in XY-path of packet (1) in a way that they disrupt packet(1) (Fig. 3. c).

Scenario 4. R10 sends two packets to each of the routers R30 and R21 as soon as it receives the packet (1); hence, they will cause disruption to packet (1) in all directions (Fig. 4. a).

Scenario 5. R10 sends two packets to R30 causing DyAD and Odd-Even to rout packets in south direction of R10 to avoid being delayed. On the other hand, R11 sends two packets to each of its south and east neighboring routers that would cause delay for packet(1) if DyAD or Odd-Even was used(Fig. 4. b).

Scenario 6. As illustrated in Fig. 4. c routers R10, R11, R21 and R31 send some packets to their neighbors making delay for packet (1) while it passes through them.

These scenarios can be divided into two categories. In the first three ones most of the network traffic is directed in XY-path of packet (1). As illustrated in Fig.5 in these scenarios, DyAD and Odd-Even avoid congestion by monitoring their neighbors and thus have less end to end packet latency. Also, DyAD has better results than Odd-Even because it exploits the low latency of deterministic routings in low traffics. The second three scenarios show distributed traffic in which disrupting packets exist in all possible directions, by which the target packet can get to its destination. These scenarios investigate the impact of low latency of deterministic routings which is the result of their simplicity in contrast to adaptive ones. As shown in Fig. 6, XY works as the best in these cases; as stated in [19] because XY has a global and long term knowledge about the traffic, it exhibits better results than the others.

7. Conclusion and Future Work

This paper used formal methods that are able to perform exhaustive verification to performance prediction on GALS NoC in the early phase of design flow. To this end, a formal model for GALS NoC was presented using high level modeling language Rebeca. The model was then used for comparison between three routing algorithms, namely XY (deterministic), Odd-Even (adaptive) and DyAD (dynamically adaptive and deterministic) with respect to the maximum end-to-end packet latency. Results of comparison are presented under two different traffic patterns and show that under distributed traffic a deterministic routing could better work. However, in a directed traffic -that is of more interest in real applications- adaptive routing algorithms are better. The routing performance results obtained

[1] We mean two packets are sent with negligible delay between two routers.

through Rebeca model checking confirm the same previously published results in simulations.

Results of such comparisons can help designers to make early decision about the parameters of the system based on the performance parameters. To have more realistic model and more precise analysis, our model can be extended by inserting more details of the system along with progress in the design flow, which we leave as future work.

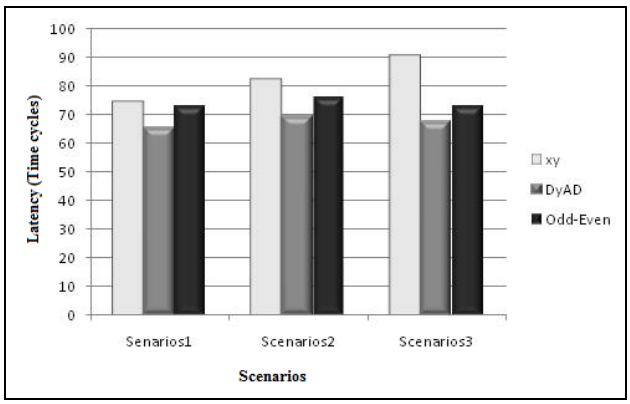

Fig. 5: Results for comparison of XY, DyAD and Odd-Even under 1-3 scenarios

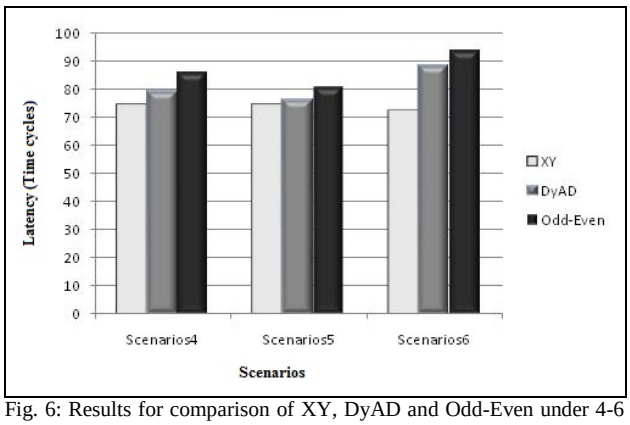

Fig. 6: Results for comparison of XY, DyAD and Odd-Even under 4-6 scenarios

References

[1] *International Technology Roadmap for Semiconductors*- ITRS 2011, http://www.itrs.net/Links/2011ITRS/Home2011.htm

[2] C. Baier, B.R. Haverkort, H. Hermanns and Joost-Pieter Katoen. "Performance Evaluation and Model Checking Join Forces." Commun. ACM, vol. 53, 2010, pp. 76-85.

[3] E. Khamespanah, Z. Sabahi Kaviani, R. Khosravi, M. Sirjani and M.J. Izadi. "TimedRebeca Schedulability and Deadlock-Freedom Analysis Using Floating-Time Transition System." in *Proc*. AGERE!, 2012, pp. 23-34.

[4] M. Sirjani, A. Movaghar, A. Shali and F. S. De Boer, "Modeling and Verification of Reactive Systems using Rebeca," Fundemental Information, vol. 63, pp. 385-410, Dec. 2004.

[5] M. Sirjani and M. M. Jaghoori. "Ten Years of Analyzing Actors: Rebeca Experience." in Formal Modeling: Actors, Open Systems, Biological Systems, 2011, pp. 20-56.

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

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

[8] A. Sen, V. Ogale and M. S. Abadir. "Predictive Runtime Verification of Multi-Processor SoCs in SystemC," *in 2008 Design Automation Con.f*, pp. 948-953.

[9] W. Dally and C. L. Seitz, "Deadlock-Free Message Routing in Multi-processor Interconnection Networks," *IEEE Trns. Computers*, vol. 36, pp. 547-553, May 1987.

[10] M. Najibi and P. A. Beerel, "*Performance Bounds of Asynchronous Circuits with Mode-Based Conditional Behavior*," in *2012 IEEE Asynchronous Circuits and Systems Conf.*, pp. 9-16.

[11] G. Madl, "Model-based Analysis of Event-driven Distributed Real-time Embedded Systems," Ph.D. dissertation, Univ. California, 2009.

[12] A. Brekling, "Modelling and Verification of MPSoC," M.Sc. dissertation, Univ. Technical University of Denmark, 2006.

[13] Y. Chenl, W. SU, P. Hsiungt, Y. Lan, Y. Hu and S. Chen. "Formal modeling and verification for network-on-chip." in *ICGCS*, 2010, pp. 299-304.

[14] N. Coste, H. Hermanns, E. Lantreibecq and W. Serwe, "Towards Performance Prediction of Compositional Models in GALS Designs," in *Proc. 2009 Computer Aided Verification Conf*., pp. 204-218.

[15] S. Foroutan, Y. Thonnart, R. Hersemeule and A. Jerraya, "Analytical Computation of Packet Latency in 2D-Mesh NoC," in *proc. 2009 Circuits and Systems and TAISA Conf*., pp. 1 – 4.

[16] C. Hewitt, "Description and Theoretical Analysis (Using Schemata) of PLANNER: A Language for Proving Theorems and Manipulating Models in a Robot," MIT Artificial Intelegence, Tech. Rep. TR-258, 1972.

[17] Rebeca Homepage, http://www.rebeca-lang.org/wiki/pmwiki.php/Examples/NOC4x4

[18] G. M. Chiu, "The odd-even turn model for adaptive routing," *IEEE Trans. Parallel and Distributed Systems*, vol. 11, pp. 729-738, Jul. 2000.

[19] J. Hu and R. Marculescu, "DyAD – Smart Routing for Networks-on-Chip," in Proc. 2004 IEEE Design Automation Conf., pp. 260-263.

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