Modeling Networking Issues of Network-on-Chip: a Coloured Petri Nets Approach

Hamid Hajabdolali Bazzaz[1], Marjan Sirjani[1,2], Ramtin Khosravi[1], Shamim Taheri[1]

[1] School of Electrical and Computer Engineering
University of Tehran
Kargar Ave., Tehran, Iran
{h.hajabdolali, rkhosravi, s.taheri}@ece.ut.ac.ir

[2] School of Computer Science
Reykjavik University
Kringlan 1, 103, Reykjavik, Iceland
msirjani@ut.ac.ir

ABSTRACT

Network-on-Chip (NoC) is proposed as a new scalable architecture to address the future design challenges of system-on-a-chip (SoC). As current verification techniques for on-chip communication algorithms are typically complicated tasks including many hardware modules and software routines, verifying the algorithms themselves is almost impossible. Having the incentive for simplifying verification of these on-chip algorithms, in this paper, we propose a detailed NoC CPN model in which key NoC networking challenges, namely network topology, switching method, and routing algorithm are considered. By this model, any desired NoC topologies, including but not limited to, mesh and k-ary n-cube can be constructed. As for switching techniques, dominant on-chip switching methods, namely, packet switching, circuit switching, and wormhole switching, are modeled. Besides, as model of a NoC switch element is highly dependent on its switch fabric type, different sorts of switching fabrics, i.e., crossbar and shared bus, are modeled in this contribution. For routing the packets between cores, a CPN version of dimension-ordered routing, dominant routing algorithm for NoC, is implemented in the switches.

Categories and Subject Descriptors

D.2.2 [**Design Tools and Techniques**]: Petri nets; B.4.4 [**Input/Output And Data Communications**]: Performance Analysis and Design Aids—*Formal models, Simulation, Verification*

General Terms

Design, Verification

Keywords

Modeling, Network-on-Chip, Coloured Petri Nets

1. INTRODUCTION

Network-on-Chip is a new architecture designed to improve the scalability of the future system-on-chips (SoC). In this architecture, instead of using a shared-medium bus, Intellectual Property (IP) cores communicate utilizing an on-chip network, i.e., network switches, placed on the chip in a structured topology. For providing an on-chip network communication, already solved computer networks networking challenges should be solved again specifically for this on-chip architecture. The most networking challenging issues include network topology, switching method and routing algorithm. Different sorts of network topologies are already proposed for NoC, among which mesh and k-ary n-cube are the most popular ones according to their regularity, symmetry and low diameter. The switching technique determines when and how switches are set to connect their input ports to output ones and the time at which message components may be transferred along these paths. [1] Most of the switching methods used in traditional computer networks are also implemented on-chip, and especially a switching method named *wormhole* switching is the most popular one for on-chip communication according to the small buffer requirement in network switches. The routing algorithm picks the path for each packet to reach its destination. Among different routing algorithms proposed for on-chip routing, *dimension-ordered* routing algorithm is the most famous one according to its best use of regularity of the topology and simplicity of implementation.

As the process of on-chip communication algorithm implementation and verification is currently a complicated task including many hardware modules and software routines, verifying the algorithm itself by implementation is almost impossible. In fact, many phases in hardware design and synthesis require a golden model in order to verify just the hardware implementation to be equivalent to the model. Therefore, applying modeling and verification techniques may be an appropriate alternative. On the other hand, an appropriate NoC model needs to cover three key issues discussed earlier; namely, network topology, switching method, and routing algorithm. The network topology model is simply a matter of the way switches and cores are connected to each other. Therefore if the model of switches and cores are independent of the topology type, then there is nothing to be modeled about the topology but the communication links. These links connect the NoC elements (switches and cores) to each other. On the other hands, Switching methods and routing algorithms are implemented in NoC switches and the switch model is highly affected by its switch fabric type; therefore for a true NoC modeling, the switch fabric model

SIMUTools 2009, Rome, Italy.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5574
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5574

should be considered, as well.

Recently, it has been proved that employing deterministic and stochastic Petri nets to model on-chip communication is an attractive method to evaluate and explore different communication aspects. In [2], it is shown that by applying these modeling techniques, it is possible to efficiently trade off modeling effort against modeling accuracy. Based on the stochastic behavior of generalized stochastic Petri nets (GSPN) [3] and coupled with the coloured Petri nets (CPN) [4], in [5][6], a new high-level net named S-net, for modeling multistage interconnection networks is proposed. In [2][7][8][9], Blume *et al.* have modeled basic NoC communication scenarios featuring different processor cores, network topologies and communication schemes. All these papers on modeling NoC by means of Petri Net and its variants, mainly aimed at performance evaluation. Therefore, mostly the stochastic Petri nets (SPN), which major usage is performance evaluation [10], is applied for modeling [2][7][9][11]. By incorporating the concept of time into SPN models these works have evaluated different performance metrics such as *bus usage percentage* [2], *average establishing time for NoC* [7], etc. These papers propose a high level model for on-chip communication and aim at modeling hardware related aspects of NoC [2][5][6][7][8][9].

In this contribution, we propose a detailed model for NoC networking concepts utilizing CPN. This model makes the verification and analysis of the algorithms and methods themselves (rather than their performance) much easier than the current techniques available for on-chip networks. The presented model considers all the aforementioned required elements for an appropriate NoC model. Specifically, by our link model, any desired NoC topology, including but not limited to, mesh and k-ary n-cube may be constructed. All the popular existing switching counterparts, namely packet switching, circuit switching, and wormhole switching are modeled too. As the switch model is highly dependent on the switch fabric type, crossbar and shared bus models are considered as the fabric types choices. Finally, the dominant on-chip routing discipline, dimension-ordered, is modeled in our switches. Recently, Blume et al in [8] proposed a similar but high-level model for NoC communication using CPN in which they have not considered several important issues such as link and switch fabric models. Also in their work, circuit switching (called line switching in their paper) is the only switching method modeled and even wormhole switching, the dominant on-chip switching method, is not modeled at all.

The remainder of this paper is organized as follows. Section 2 reviews the basic concepts of CPN. Section 3 reveals the different networking concepts of NoC and our related models in CPN. Analysis and results of our simulations are presented in section 4 and section 5 concludes the paper.

2. BRIEF REVIEW OF COLOURED PETRI NETS

Coloured Petri nets (CPN) has been developed in 1980 by K. Jensen in his PhD thesis to expand the modeling possibilities of classical Petri nets. A complete overview of the modeling possibilities with CPN is not in the scope of this paper. However, here we briefly bring up the basic features of the CPN used in our work. For a thorough overview, see e.g., [11].

Petri nets consist of so-called *places*, *arcs* and *transitions*. Places, depicted as ellipses in the graphical representation, model the states of system components while transitions shown as rectangles are used to change the state of the system. Places can be marked by tokens which are shown by circles (Fig. 1). The high level data types of programming languages are called *colorset* in CPN and tokens represent high level data structures (variables), and their values (the data stored in them) are shown in a rectangle attached to the token's circle. Tokens, as well as places, are always associated with a colorset and a place may only contain tokens of the same colorset as its own. The colorset of a place is attached below the place in the graphical representation. Transitions and places are connected via arcs. A transition is *enabled* for being *fired*, once all the input places connected to it are marked and its *guard* condition is satisfied. Guard conditions are encased in brackets and written above the transition. When a transition is fired, one token from its every input place is deleted and a token is added to all of its output places. Transitions can access the data stored in tokens by mapping tokens to variables. A Transition may also have a *transfer function*, which is a code segment that can access the mapped variables of the transition and modify them. Transfer function consists of the definition of the input and output variables and also the commands which should be carried out (*action*). The transfer function is attached below the transition in the graphical representation. If more than one transition is enabled, one of them is randomly chosen to be fired.

Currently, many tools are available for Petri Net modeling [12]. In this paper, we have used *CPNtools* software for modeling, simulating and analyzing the NoC architecture [13]. This tool is available for free for academic purposes. CPNtools has a simulator and a rich graphical user interface for composition of CPN models. One of the most useful features of CPNtools is its support for hierarchical designs, which facilitates the reuse of some parts of the design and simplifies handling of large models.

3. MODELING NOC

Our NoC model is divided into different parts; namely, communication link, switch fabric, and different switching methods models. As the dimension ordered routing is locally performed in the switches and is highly closed to the switching method, the related CPN model is presented with the switching method model together. The presented models may be applied to any sorts of topology in which every switch can be addressed by a pair of X and Y coordination. Now, before presenting our CPN NoC model, we are to state the related data structure.

3.1 Data Structure Model

The CPN data structures (colorsets) and variables which are used in this paper are explained in this section. Table 1 shows the related data structure. *AddressX*, *AddressY*, and *Address* are used to describe the address of the cores which is used in *AddressHeader* declaration. The *PORT* is defined as an enumeration, which is used in the *FWTableEntry*.

FWTableEntry is an entry in the switch forwarding table for circuit switching method, which is an input port to output port mapping. *PSPacket*, *CSPacket*, and *WSPacket* define the packet switching, circuit switching and wormhole switching packet data structures, respectively. *CSPacket-*

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5574
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5574

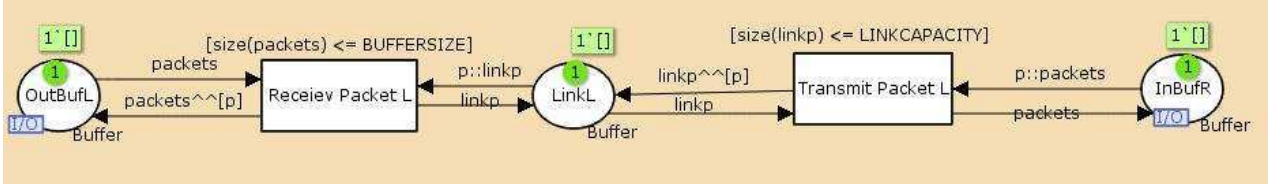

Figure 1: The communication link model in CPN

Table 1: The datatypes used to model NoC

```
val TopSize = 4
val BUFFERSIZE = 3
val LINKCAPACITY = 1

colset AddressX = int with 0..TopSize;
colset AddressY = int with 0..TopSize;
colset Address = record x: AddressX * y: AddressY;
colset AddressHeader = record src: Address * dst: Address;
colset Payload = STRING;
colset Port = with LEFT | RIGHT | UP | DOWN | CORE | NULL;
colset FWTableEntry = record inport: Port * outport: Port;
colset ForwardingTable = FWTableEntry;
colset PSPacket = record header: AddressHeader * data: Payload;
colset CSPacketType = record head: BOOL * tail: BOOL * ack: BOOL;
colset CSPacket = record ptype: CSPacketType * address: AddressHeader * data: Payload;
colset LabeledCSPacket = product CSPacket * Port * Port;
colset WSPacket = CSPacket;
```

Type is used to recognize different packets in the circuit and wormhole switching.

TopSize defines the size of mesh topology, that is the number of switches in X and Y dimensions. The *BUFFERSIZE* and *LINKCAPACITY* are used to define the capacity of the switch buffers and the number of in-flight packets of the links, respectively. Also note that the buffers in the model are of the *Buffer* colorset, which is a list of PSPacket in the case of packet switching, CSPacket in the case of circuit switching and WSPacket in the case of wormhole switching (This selection is to avoid using union structure of CPN, which makes the arc inspirations and transition guards much more complex). Table 2 shows the variables declared for our models.

Table 2: The variables declared to model NoC

```
var p: PSPacket;
var csp: CSPacket;
var packets, linkp, inpackets, packetstream,
 inflightpackets: Buffer;
var lcsp: LabeledCSPacket;
var switchaddr: Address;
var outport: Port;
var destx: AddressX;
var desty: AddressY;
var fwe, fwer, fwed, fwel, fwec,
 fweu: FWTableEntry;
```

3.2 Communication Link Model

The on-chip physical link provides the physical communication medium to connect neighbor switches to each other. The related model in CPN is depicted in Figure 1 Note that the link is full-duplex and only the side related to the right-to-left communication is shown in the Figure for the sake of simpilicity (the left-to-right model is similar). The model consists of three buffer places. One of them is related to the left side output buffer of the link (*OutBufL*). The right side input buffer of the link (*InBufR*) is also considered. *LinkL* place is to hold the in-flight packets between the left side output buffer and the right side input one. *Receive Packet L* transition is to transmit packets from output buffer of the left side to the link and the *Transmit Packet L* transition is to receive and buffer packets at the right side of the

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5574
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5574

link. Although in computer networks packets are sent over links regardless of the buffer capacity at the other side, the on-chip communication requires explicit buffer availability at the other side to send a packet. This property is indeed related to the link layer reliability of the on-chip communications, which is due to lack of higher reliable communication layers (something like the transport layer in the computer networks communication protocol stack). This fact explains the reason behind specific guard [size(packets) $\leq$ BUFFERSIZE] provided for the left transition. The guard provided for the other transition is for the fact that in-flight packets number in the link is actually physically limited.

When the *InBufR* place contains some packets, the transition *Transmit Packet L* is enabled for firing. After this transition is fired, the first packet in the buffer is transmitted to the link and is added to the end of the packet list in the *LinkL* place. When *Receive Packet L* transition is fired, the first packet in the link is arrived at the other side and is added to the end of the packet list in the *OutBufL* place.

The port-type tags of the buffer places (such as *OutBufL* and *InBufR*) are for the hierarchical design. This indicates that the token flow direction from/into these places in the link model is bi-directional.

3.3 Switch Fabric Model

The switching fabric is the hardware logic by which the switch input ports are connected to the output ports for forwarding packets to their respective destinations. In the shared bus schema, every packet arriving at one of the switch input ports have to lock the shared bus and being transfered to it in order to be forwarded to the appropriate output port. If the bus is already locked, the packet is buffered in the input buffer, waiting to acquire the shared bus some time later. It is clear that in this schema, the shared bus is the bottleneck of switching speed performance. On the other hand, in the crossbar schema, every single input port has a distinctive wiring path to every output port; hence several packets may be forwarded from different input ports to different output ports at the same time.

Figure 2 shows the shared bus fabric model. The switch has input and output interfaces (places in Petri nets) in right, down, left, and up directions. There is also an input and output place related to the core connected directly to the switch. This is due to the selected mesh topology for the NoC structure, in which every switch is directly connected to a core. The *Address* place contains the switch address in the topology which is used in the routing algorithm. The *SharedBus* and *BusLock* places are to model the bus fabric. There is a dedicated transition for forwarding a packet from every input port to the shared bus and also from the shared bus to every output port. Assume that a packet arrives from the up input port which should be forwarded to the left output interface. To acquire the shared bus, the transition From *Up* has to get the bus lock token which is in the *BusLock* place. This fact guaranties that only one of the input places has the shared bus at a time. After the *To Left* transition is fired, lock token of this bus is put back and the bus is free for acquiring. The guard written on the *To Left* transition is to indicate that the packet should be forwarded to this output interface, and is related to the routing algorithm which is explained later. This also checks that the output buffer has the capacity for holding a new packet. To make the model less complicated, among the arcs between *Address* and *BusLock* places and the transitions, only the one related to the explained up to left path is shown in this Figure. Indeed, there should be a bidirectional arc between every transition related to output forwarding (i.e., *To Left*, *To Up*, ...) and also there is an arc to *BusLock* place from these transitions and the transitions related to input forwarding (i.e., *From Left*, *From Up*, Ě) requires an input arc from *BusLock* place. Pay attention that every transition related to output forwarding requires a guard, and in the Figure only the one related to the *To Left* transition is shown. Figure 3 shows the crossbar switch fabric model. In the crossbar model, there is a single transition for every possible forwarding path, i.e., for every input to output combination. To simplify understanding of the Figure, only the transitions related to forwarding packets to the left interface is shown. There are four similar transitions for every other input pair. The transition guards are to assure appropriate output interface selection and also buffer availability at the output port.

Table 3: The dimension-ordered routing algorithm pseudo code

```
if dest.x > switch.x
  then forward the packet to the right interface
else if dest.x < switch.x
  then forward the packet to the left interface
else if dest.x = switch.x and dest.y < switch.y
  then forward the packet to the up interface
else if dest.x = switch.x and dest.y > switch.y
  then forward the packet to the down interface
else if dest.x = switch.x and dest.y = switch.y
  then forward the packet to the core interface
```

3.4 Routing Algorithm and Switching Method Models

The switching methods proposed for on-chip communication are inspired by those in computer networks and are customized for it. In this work, we have modeled packet switching, circuit switching and wormhole switching. As the switching method and routing algorithm work very closely together to forward packets to their destinations, we explain them together in this part.

The dimension-ordered routing algorithm, which is the most well-known routing algorithm, is used in our models. In this algorithm, each packet is routed to its destination in the X coordinate first and then to the Y coordinate. The related pseudo code is depicted in Table 3. As the conditions are independent of each other, the if-clause conditions can simply be used for the transition guards similar to the left forwarding guard shown in Table 3.

In the packet switching method, every packet contains the destination address header to be routed independently to its destination. This makes the modeling and implementation much easier than other methods. Indeed, in Figures 2 and 3, this switching method is considered and that is why the packet type is not examined and all the packets are behaved in the same way.

In circuit switching, a virtual circuit is set up for every flow on the chip. Hence, before sending data packets, the sender

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5574
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5574

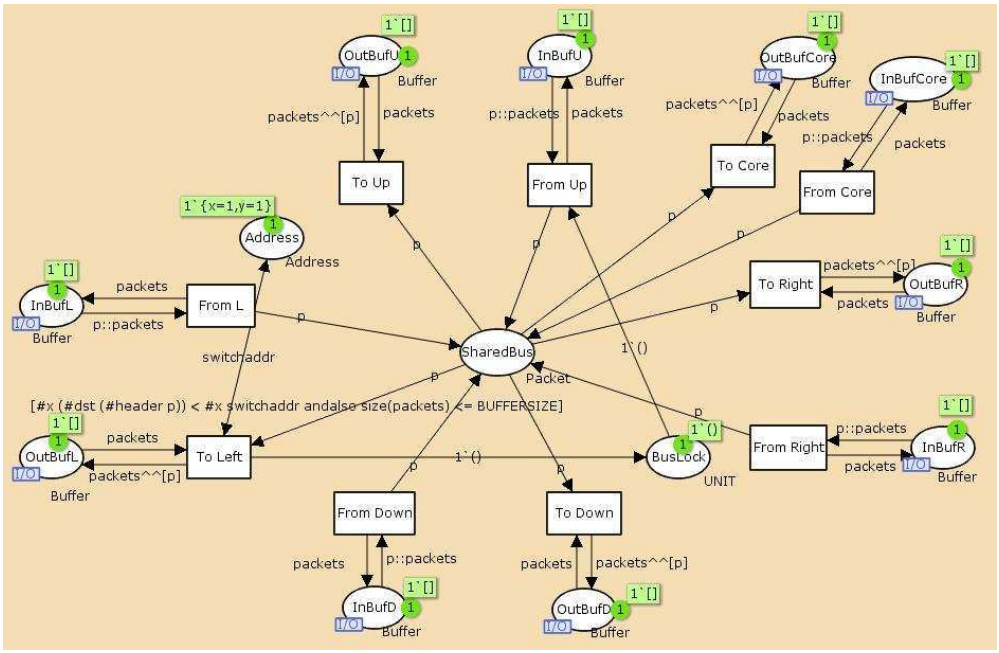

Figure 2: The shared bus switch fabric model in CPN

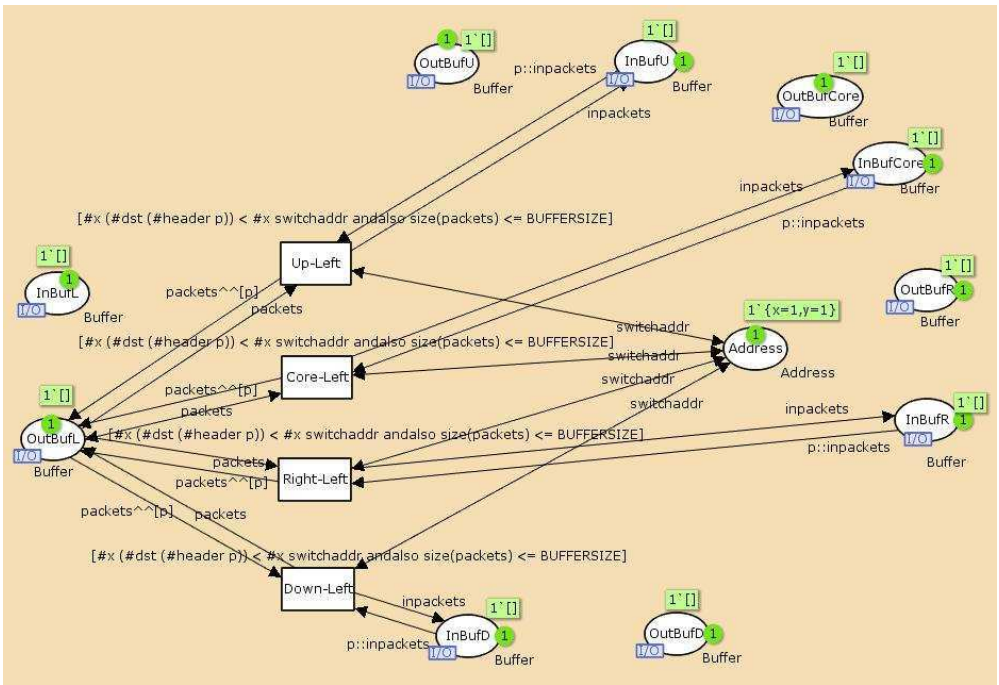

Figure 3: The crossbar bus switch fabric model in CPN

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5574
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5574

transmits a special packet which is called *routing probe*. This special packet, which we call *head packet*, is to set up the circuit in every switch and update their forwarding tables prior to data sending phase. This situation speeds up the forwarding of data packets as there is no need to run the routing function and decide the appropriate output interface. Once the source sent the head packet, it waits for the arrival of the ack packet which is sent by the destination after receiving the head packet. When the source does not have any more data to send, it tears down the virtual circuit by sending a *tail* packet. All the forwarding table entries related to this flow are deleted once the tail packet traverses the path from the source to the destination. Figure 4 depicts the circuit switching model related to the switching behavior for a header packet arrived from the left input interface of the switch which should be forwarded to the down output port. The *SwitchForwardingTable* place contains the forwarding table entries related to each input port. Initially, as no circuit exists, all the output ports of the forwarding table entries are set to *NULL*. When the *Head Packet L* transition is fired, the code segment related to this action makes a *LabeledCSPacket* from the corresponding *CSPacket* and places the packet to the *WaitForOutputPort* place. This labeled packet contains the input and output port related to this packet, and is used to forward the packet and also update the forwarding table.

The *LabelPacket* transition reads all the forwarding table entries of the switch and its guard, *checkoutputAvailability*, checks that no input port is already mapped to the output port to which this packet should be forwarded. So, the packet waits in the *WaitForOutputPort* place until the related output port is free. When the packet is sent to the *Forward* place, the output port is available for forwarding the flow packets. The guard on *ToDown* transition selects only the packets which are labeled to be forwarded to the down output interface. Once this transition is fired, the forwarding entry related to the left input port (which was omitted from the table in *Head Packet L* transition) is added to the table for the other packets of the flow. As the *Forward* place and *ToDown* transition is used for every kind of packets, the arc inscription depends on the type of the packet. In the case of a head packet and data packets, the forwarding entry related to the input port is added (inport=LEFT, outport=DOWN in this case). In the case of a tail packet the output port is set to *NULL* in the related entry, and in the case of an ack packet, there is no need to put anything in the table. Note that the forwarding table is omitted whenever a packet arrives and is put back when the packet leaves, so that the packets behind it in the input buffer can not enter before it is forwarded, i.e., the forwarding entry is also used as a lock to gain in-order delivery.

Figure 5 shows the similar scenario for data, tail, and ack packets. The *Circuit Exist L* transition is for the case of a data packet (that is a non-head, non-tail and non-ack packet). The guard provided for the transition is to check that the output port is already set for the input port. The input packet is labeled using the forwarding entry found in the forwarding table and is put in the *Forward* place to be sent to the related output port exactly in the same way as the head packet. Notice that there is no need for the switch address, as the output port is found only using forwarding table. The *Tail Packet L* transition is for the tail packet, which is similar to the data packets case (indeed, these transitions can be joined, but here, we preferred to separate them to make the concept simpler). Besides, note that in the case of a tail packet, the forwarding table entry should be deleted which is done using the inscription provided on the *ToDown* to *SwithForwardingTable* arc. An ack packet should be traversed in the reverse path of the head packet. This explains the guard expression in the *ACK Packet L* transition.

It is important to notice that in the circuit switching method, there is a need to model the sender and receiver sides of the communication, i.e., cores. The sender should send a head packet and then wait for the ack packet from the receiver. Once the ack packet is received, it sends the data packets and at last closes the connection using the tail packet. Our sender and receiver models related to the simulations are explained in the next section.

In the wormhole switching schema, the sender divides every packet to small units named flits. As only a few numbers of *flits* (one to three) are buffered in every switch, the buffer requirement per switch is reduced. This is an important gain for an on-chip communication. Also, the sender does not wait for the ack arrival to start sending the data packets. Instead, the data packets are immediately sent after the head packet, which helps to reduce the total delay for delivering packets of a flow. The switching method is almost the same as the circuit switching case; there is no need for the ack packet support for the switches in this case and also the sender and receiver side of the communication are different from the circuit switching case.

4. SIMULATION RESULTS

The Simulations are performed on a 4x4 mesh topology topology setup shown partially in Figure 6. A bi-directional communication link connects each pair of neighboring switches and corresponding input/output port buffers of each switch are connected to the associated links as shown in the Figure. Each switch is directly connected to an IP core via the link of its associated core ports. Different sender-receiver CPN models (core models) are considered in the experiments. A simple sample core implementation for circuit switching disciple is shown in Figure 7. The upper half of the model represents the sender side while the lower half one corresponds to the receiver side. After Sending a packet to a destination (core 2-2 in this case), the sender waits for arrival of an ack in *WaitForACK* place. Once the ack is received, the sender generates the data packets and the tail packet and buffers them in its output buffer. The packets are then forwarded through the on-chip intermediate switches to the receiver. The receiver side, on the other hand, initially waits for arrival of a header packet (in *WaitForHead* place). Once the head packet is received, the receiver sends an ack packet back to the sender, acknowledging the possibility of receiving a bunch of data packets, and then changes its status to *WaitForData*. The receiver is informed about the end of data stream packets by receiving the tail packet; it then switches back to *WaitForHead* by firing the *ReceiveTail* transition. It is worth noting that after the sender receives the ack packet, it doesn't work in parallel with the receiver. The sender sends his data by his own rate and ends it by the tail packet. While on the other side, the receiver buffers and processes the data packets independently of the sender rate as soon as they arrive and tears down the connection (i.e., releases its allocated buffers and variables) as the tail packet is received. One may note that this separation of sender-receiver rates

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5574
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5574

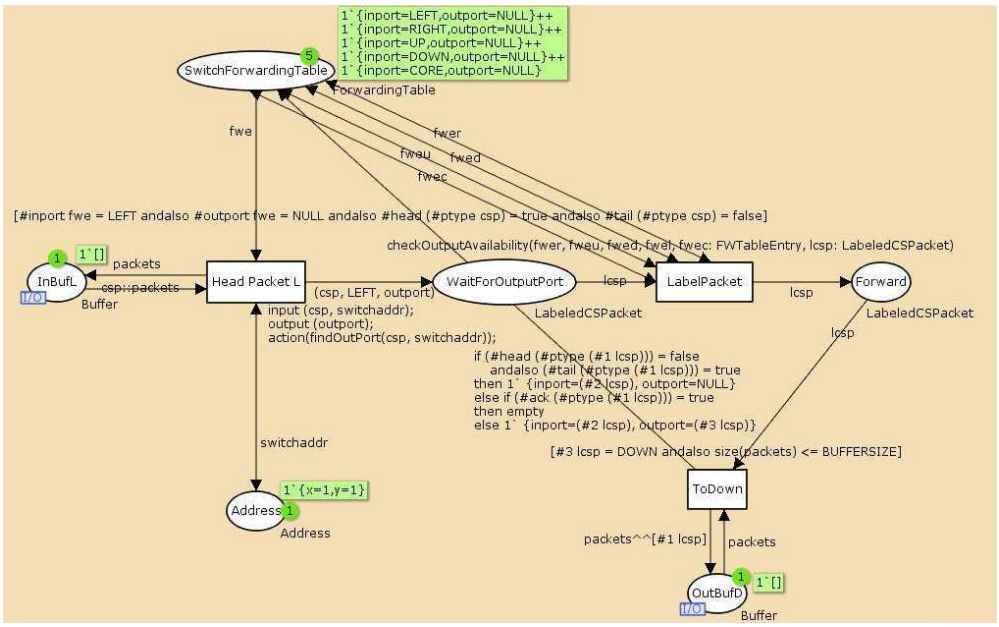

Figure 4: The circuit switching method model for a header packet in CPN

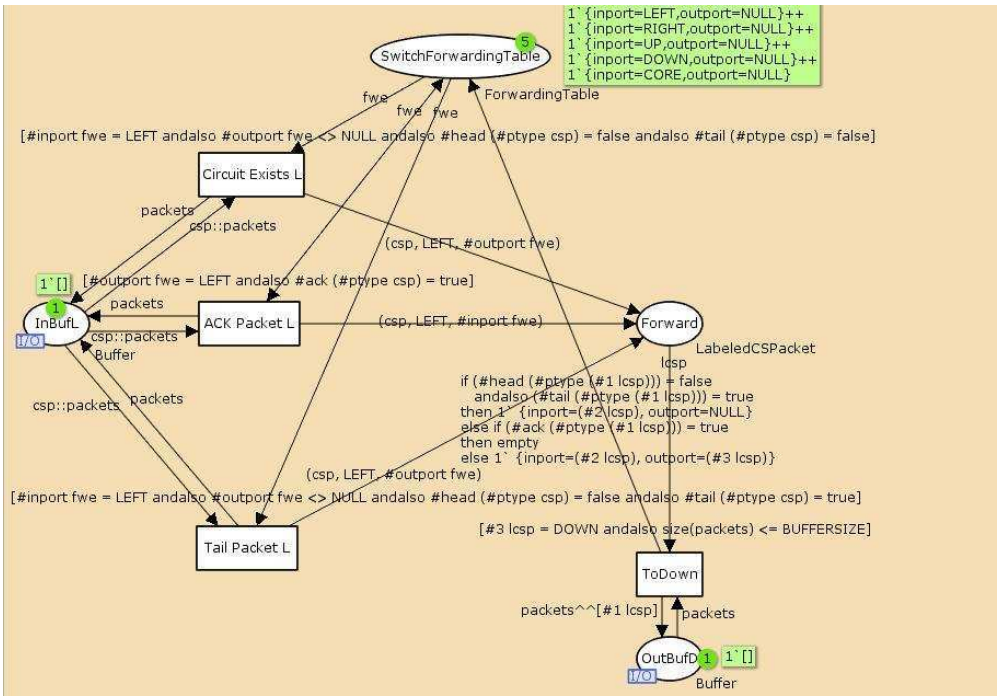

Figure 5: The circuit switching method model for data, tail, and ack packets in CPN

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5574
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5574

is solely a result of wormhole switching.

Different communication scenarios are considered in the experiments. In one scenario, a subset of cores send packets to different predetermined set of other cores; while in another case every sender transmits a stream of packets to every other core in the topology. Tokens (packets) are generated using the transfer function feature of the transitions in CPN.

By the help of the *Monitoring* and *State Space* tool boxes of CPNtools, verification of correctness of different aspects of the on-chip networking algorithm is possible. Specifically, by simulating diverse scenarios and varying simulation settings such as communicating pairs, communication patterns and switches and links buffer size, it is observed that corresponding on-chip switching methods and routing algorithms are *dead-lock* free. Quantitative measures of the system, such as average delay for receiving a packet (number of transitions), average queue size or switches load distribution during the simulation are also of interest. Due to space constraints, here we only present the results for switches load distribution measure. The switch load percentage is defined as the ratio of the sum of the number of tokens in its places out of the total number of tokens in all of the places during the simulation. Figure 8 shows the topology switches load distribution for the case of uniform traffic generation between every two cores of the topology. The x and y dimensions of the graph corresponds to the location of the switches on the board and the z dimension shows the related switch load percentage. The simulation results show that the obtained graph is a regular hotspot with its maximum centered at its middle. The reason behind this is that the mesh topology is symmetric to its center; hence, most of the packets are routed through central switches on the path to their destination.

Another interesting scenario to study is comparison of different switching techniques and switch fabric types. Moreover with the help of the proposed CPN model, it is even possible to have switches of different fabric and switching model together on a single topology. By this setup more complicated scenarios are examined and interaction of different switching techniques together is studied. The results of these studies will be presented in our future works according to the lack of space in this paper.

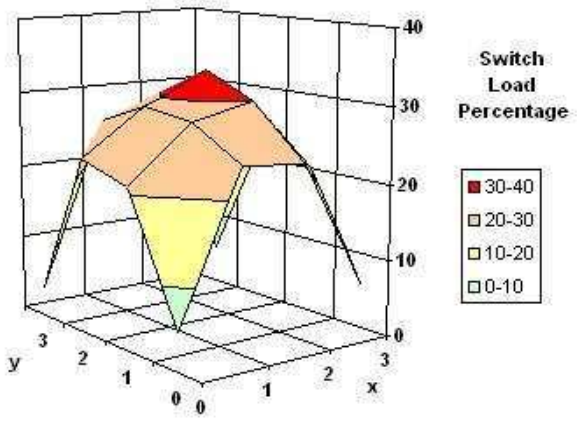

Figure 8: Switch load distribution for the case of uniform traffic generation between cores. The x and y dimensions of the graph corresponds to the location of the switches on the board.

5. CONCLUSIONS

The interest in Network-on-Chip (NoC) as a new scalable architecture for future system-on-a-chip (SoC) designs is increasing rapidly. Hence, the need to provide more customizable and easier verification techniques in an early stage of design process is a critical issue. In this paper, applying the CPN, the networking issues of NoC are modeled, simulated, and analyzed. As the modeling is done in a high level parallel schema, the details of the underlying algorithms can be verified by different simulation scenarios. In addition, by applying the proposed CPN model of NoC, one may vary the simulation scenarios and customize them much more easily than current existing techniques for NoC simulations.

6. REFERENCES

[1] Jose Duato, Sudhakar Yalamanchili, and Lionel Ni. *Interconnection Networks: An Engineering Approach.* IEEE Computer Society Press, 1997.

[2] Holger Blume, Thorsten von Sydow, and Tobias G. Noll. *Performance Analysis of SoC Communication by Application of Deterministic and Stochastic Petri Nets*, pages 484–493. Springer Berlin / Heidelberg, 2004.

[3] G. Chiola, M. A. Marsan, G. Balbo, and G. Conte. Generalized stochastic petri nets: A definition at the net level and its implications. *IEEE Trans. Softw. Eng.*, 19(2):89–107, 1993.

[4] Kurt Jensen. Coloured petri nets and the invariant-method. *Theoreiical Computer Science 14*, pages 317–336, 1981.

[5] N. Chaki and S. Bhattacharya. Performance analysis of multistage interconnection networks with a new high-level net model. *J. Syst. Archit.*, 52(1):56–70, 2006.

[6] S. Bhattacharya N. Chaki. Modeling and analysis of interconnection networks using high level net. In *15th IASTED International Conference on Modelling and Simulation*, Innsbruck, Austria, February 1996.

[7] H. Blume, T. von Sydow, D. Becker, and T. G. Noll. Application of deterministic and stochastic Petri-Nets for performance modeling of NoC architectures. *J. Syst. Archit.*, 53(8):466–476, 2007.

[8] J. Schleifer H. Blume, T. von Sydow and T.G. Noll. Petri net based moding of communication in systems on chip. Book chapter in Vedran Kordic (ed.) Petri Net, Theory and Application. I-Tech Education and Publishing, Vienna, Austria, February 2008.

[9] H. Blume, T. von Sydow, D. Becker, and T.G. Noll. *Modeling NoC Architectures by Means of Deterministic and Stochastic Petri Nets*, pages 374–383. Springer, Berlin, 2005.

[10] G. Ciardo, R. German, and C. Lindemann. A characterization of the stochastic process underlying a stochastic petri net. *IEEE Trans. Softw. Eng.*, 20(7):506–515, 1994.

[11] Christoph Lindemann. *Performance Modelling with*

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5574
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5574

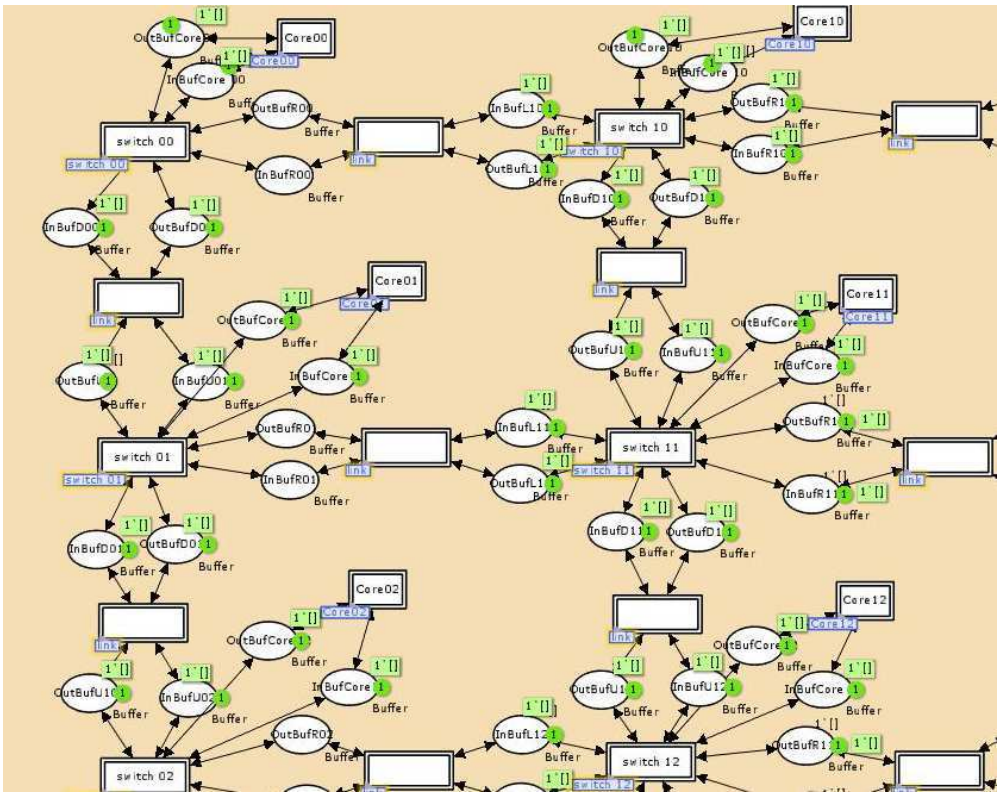

Figure 6: A part of the 4x4 mesh topology of simulations

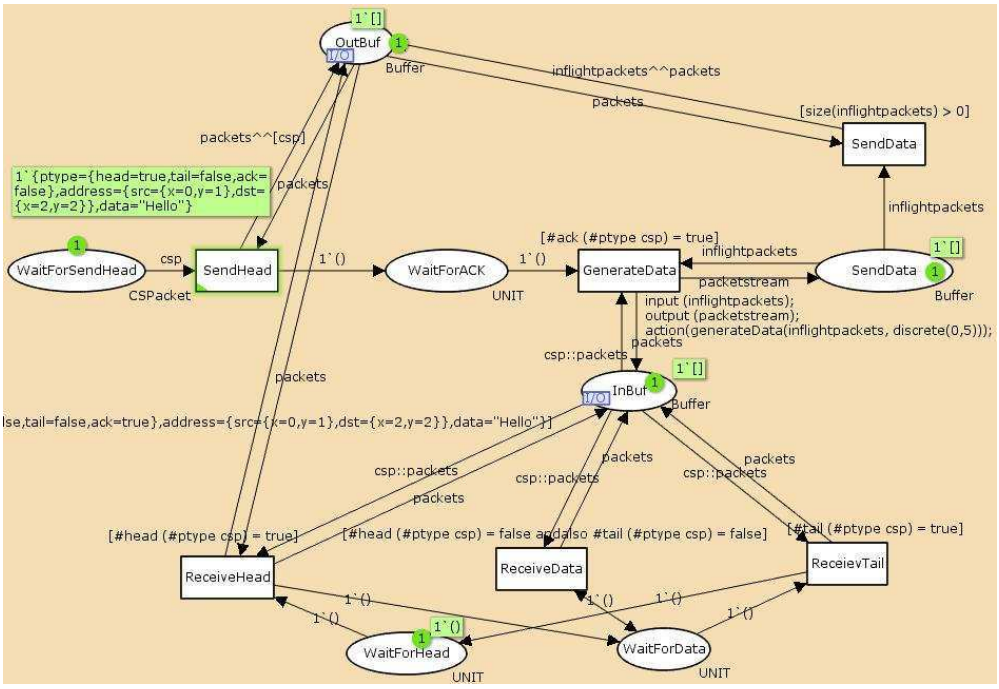

Figure 7: A simple sender-receiver (core) model

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5574
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5574

Deterministic and Stochostic Petri Nets. John Wiley; Sons, Inc., 1998.

[12] Petri nets world. http://www.informatik.uni-hamburg.de/TGI/PetriNets/.

[13] A. Vinter Ratzer et al. CPN tools for editing, simulating, and analysing coloured petri nets. In *Proceedings of the 24th International Conference on Applications and Theory of Petri Nets (ICATPN 2003)*, pages 450–462, Eindhoven, The Netherlands, June 2003. Springer-Verlag.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5574
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5574