

Formal Analysis of a Fault-Tolerant Routing Algorithm for a Network-on-Chip

Zhen Zhang, Wendelin Serwe, Jian Wu, Tomohiro Yoneda, Hao Zheng, Chris Myers

▶ **To cite this version:**

Zhen Zhang, Wendelin Serwe, Jian Wu, Tomohiro Yoneda, Hao Zheng, et al.. Formal Analysis of a Fault-Tolerant Routing Algorithm for a Network-on-Chip. 19th International Conference on Formal Methods for Industrial Critical Systems, FMICS 2014, Sep 2014, Florence, Italy. pp.48-62, 10.1007/978-3-319-10702-8_4 . hal-01064829

HAL Id: hal-01064829

https://inria.hal.science/hal-01064829

Submitted on 17 Sep 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Formal Analysis of a Fault-Tolerant Routing Algorithm for a Network-on-Chip*

Zhen Zhang[1], Wendelin Serwe[2], Jian Wu[3],
Tomohiro Yoneda[4], Hao Zheng[5], and Chris Myers[1]

[1] Dept. of Elec. & Comp. Eng., Univ. of Utah, Salt Lake City, UT, USA
zhen.zhang@utah.edu, myers@ece.utah.edu,
[2] INRIA & Univ. of Grenoble, LIG, Grenoble, France
Wendelin.Serwe@inria.fr
[3] Marvell Technology Group Ltd., Santa Clara, CA, USA
jianwu@marvell.com
[4] National Institute of Informatics, Tokyo, Japan
yoneda@nii.ac.jp
[5] Dept. of Comp. Sci. and Eng., Univ. of S. Florida, Tampa, FL, USA
zheng@cse.usf.edu

Abstract. A fault-tolerant routing algorithm in Network-on-Chip architectures provides adaptivity for on-chip communications. Adding fault-tolerance adaptivity to a routing algorithm increases its design complexity and makes it prone to deadlock and other problems if improperly implemented. Formal verification techniques are needed to check the correctness of the design. This paper performs formal analysis on an extension of the link-fault tolerant Network-on-Chip architecture introduced by Wu *et al* that supports multiflit wormhole routing. This paper describes several lessons learned during the process of constructing a formal model of this routing architecture. Finally, this paper presents how the deadlock freedom and tolerance to a single-link fault is verified for a two-by-two mesh version of this routing architecture.

Keywords: LNT, process algebra, fault-tolerant routing, Network-on-Chip, formal verification

1 Introduction

Cyber-physical systems (CPS) nowadays have ubiquitous applications in many safety critical areas such as avionics, traffic control, robust medical devices, etc. As an example, the automotive industry makes active use of CPS: modern vehicles can have up to 80 *electronic control units* (ECUs), which control and operate everything from the engine and breaks to door locks and electric windows. Currently, each ECU is statically tied to its specific sensors and actuators. This

* This work is supported by the National Science Foundation under Grants CNS-0930510 and CNS-0930225. Part of this work was performed during a visit of the first author at INRIA Grenoble Rhône-Alpes.

means that processing power between different ECUs cannot be shared, which degrades the performance of the chip due to imbalanced workload on each ECU. More importantly, this structure is susceptible to faults in that if an ECU fails, it causes a malfunction in the corresponding sensor and/or actuator. With the advances in semiconductor technology, it is now possible to have multiple cores on a single chip which communicate using a *Network-on-Chip* (NoC) paradigm. A NoC approach allows flexible mapping between ECUs and sensors/actuators, which makes it possible for ECUs to share processing power and tolerate faults by having spare units. Some example fault-tolerant NoC architectures currently being developed include those described in [1] and [2].

This paper presents the verification of a NoC architecture that supports the link-fault tolerant routing algorithm [3] extended to a multiflit wormhole routing setting. In particular, deadlock freedom and single link-fault tolerance are formally verified using the CADP toolbox. This paper also presents several key lessons that are learned during the evolution of the model of the NoC architecture. Finally, this paper describes several remaining challenges to the verification of this and similar systems.

This paper is organized as follows. Section 2 surveys related work. Section 3 describes the extended NoC architecture and routing algorithm. Section 4 presents several case studies of the process that led to the final NoC model. Section 5 presents verification results for deadlock freedom and the single-link fault tolerance property. Section 6 discusses the insights obtained from using model checking in the design of the NoC behavior and some future research directions.

2 Related Work

A fully functional NoC system has to be fault-tolerant and free of deadlocks. A variety of approaches have been proposed for fault-tolerant NoC routing. One approach is to use a reconfigurable routing table in which pre-computed routes are stored to avoid faulty links [4]. This method, however, is not adaptive, so it can only avoid permanent faults. An example of a dynamic faulty link detection mechanism is described in [5], but this method only avoids deadlocks rather than ensuring they cannot occur. The Glass/Ni fault-tolerant routing algorithm, on the other hand, guarantees deadlock freedom by disallowing certain turns in the network [6], so that communication cycles cannot occur. This algorithm, however, uses the *node-fault model*, where a fault in an incoming link is interpreted as the complete node failing. Not only does this mean losing the ability to route to an otherwise functional node, but if the node does not actually stop operating, it can potentially introduce deadlock in the network. A modified version proposed in [7] achieves one link-fault tolerance by introducing a mechanism to forward link fault locations to a neighboring routing node allowing for a route selection that avoids the faulty link. This fault forwarding method though can still result in a deadlock at the edges of the mesh network, so in these cases, it must revert to the node-fault model. An improvement proposed in [3] is capable of handling link faults anywhere in the network. Potential deadlock is avoided by

allowing a router to drop a packet to prevent the occurrence of a communication cycle, and it is an extended version of this algorithm that this paper attempts to formally verify.

Concerning NoC verification, [8] proposed GeNoC (*Generic Network-on-Chip*), a formal NoC model implemented in the ACL2 theorem prover. Its extension in [9] verifies a non-minimal adaptive routing algorithm. These techniques, however, require user assistance on writing proof obligations. On the other hand, to facilitate the use of model checking techniques, automatic translations are developed from the asynchronous hardware description language CHP (*Communicating Hardware Processes*) to networks of automata [10] and to the process algebraic language LOTOS [11,12]. The latter approach is applied to verify two complex asynchronous designs, one of which is an input controller of an asynchronous NoC [13] that implements a deadlock-free routing algorithm based on the odd-even turn model [14]. However, this NoC does not handle failures.

3 Network-on-Chip Architecture and Routing Algorithm

Figure 1 shows an architecture for a two-by-two mesh composed of four corner routing nodes, all with a similar structure. This architecture implements an extended version of the routing algorithm described in [3]. The original algorithm assumed single-flit packets and that each node could route only a single packet at a time, while this modified architecture allows each node to potentially have multiple multi-flit packets in flight at a time. For example, node 00 may be routing a packet from node 01 to node 10, while simultaneously routing a packet from node 10 to node 01. These extensions though complicate the algorithm, so it is desirable to be able to prove that this extended architecture is still deadlock-free and continues to achieve fault tolerance to a single-link failure, which is the goal of this paper.

The routing algorithm works as follows. Each node communicates with its corresponding *processing element* (PE), and when a PE xy wishes to send a packet to another node $x'y'$, it injects that packet into the network via its router (r_PE_xy). Based upon the intended destination of the packet, the router determines a direction to forward the packet. To guarantee deadlock freedom, the routing algorithm disallows certain turns in the network. Namely, a packet that is moving north in the network is not allowed to turn to the west, and a packet moving east in the network is not allowed to turn to the south. Hence, in order to avoid "illegal turns", each router sends packets south and west, as needed, before sending them north and east. After selecting a direction, the router attempts to communicate with the arbiter in charge of the desired link. At this point, one of three things can occur. First, the link may be busy, and the router must wait its turn to use the link. Second, the link may be faulty, and the router is instructed to find an alternate route. Finally, the link may be free, and the arbiter may non-deterministically select to communicate with this router over any other routers that may be trying to obtain this link. The arbiter then forwards the packet one flit at a time to the succeeding router (i.e., the router the

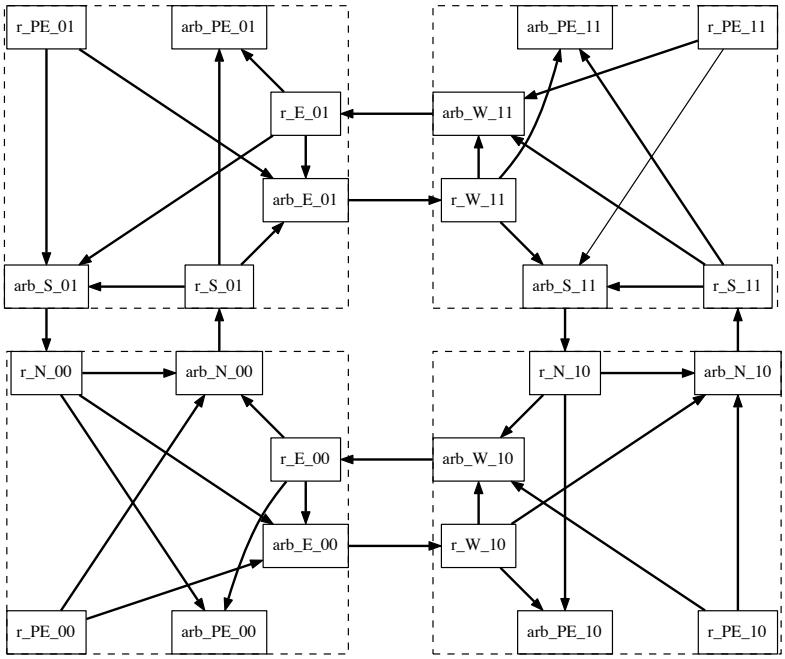


Fig. 1. Architecture of the four routing nodes in a two-by-two mesh.

output of the arbiter is connected to), which then executes the same algorithm. Once a packet reaches its destination $x'y'$, the packet is absorbed by the arbiter connected to its PE (arb_PE_$x'y'$).

Assuming there is at most one link-fault, an alternate route always exists, but it may require an illegal turn. For example, assume that node 10 wishes to send a packet to node 01. In this case, a west then north route is desired, but let us assume that arb_W_10 reports a fault on its link to r_E_00. In this case, r_PE_10 must communicate with arb_N_10 instead. Once the packet reaches r_S_11, this router must make the illegal turn and route the packet west through arb_W_11. However, arb_W_11 may be busy routing a packet from node 11 to node 00. This packet though may be blocked because arb_S_01 is busy routing a packet from node 01 to node 10. Similarly, this packet may be blocked because arb_E_00 is busy routing a packet from node 00 to node 11. Finally, this packet is blocked because arb_N_10 is busy due to the packet from node 10 to node 01. Therefore, there is a communication cycle causing a deadlock In this case, arb_W_11 sends a *negative acknowledgement* to r_S_11 to tell this router to drop the incoming packet, which removes the communication cycle and the potential for deadlock.

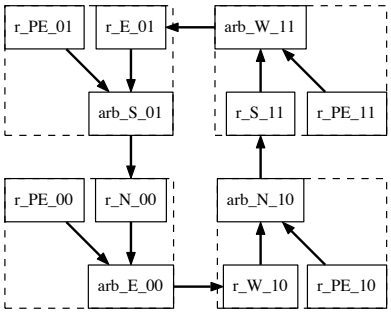


Fig. 2. A counterclockwise routing model.

4 Formal Models of NoCs

This section describes the challenges in developing a formal model of the two-by-two mesh shown in Figure 1. Our initial informal model of the two-by-two mesh uses asynchronous channels implemented in a VHDL package [15] which had to be translated into a formal model in the process algebraic language LNT [16] to enable verification using the CADP toolbox [17].

4.1 One Direction Routing

The first model developed and verified is the simple one direction routing model shown in Figure 2. It is advantageous to construct a model with one complete cycle consisting of partial components from each node, since the model is simple enough for testing asynchronous communications between any two components. Also, the resultant state space is manageable, which enables the efficient checking for deadlock and packet loss without having to abstract the model. Since this model only has the counterclockwise routing direction, there are no alternative routes available, avoiding the need to model route forwarding computation in each router. Having only the counterclockwise routing direction also forces the north-to-west illegal turn to occur on the northeast node. In this first model, each PE router only generates one single-flit packet destined to the node in its diagonal direction. For example, the PE connected to node 01 sends a packet to node 10. After emitting one packet, each PE router becomes inactive. No components absorb any packets and it is assumed that no link fault exists in the network. The expected behavior is that the packet from node 10 to node 01 gets dropped due to deadlock avoidance, and the remaining three packets keep cycling through nodes in the network forever, and no deadlock exists.

The arbiter, arb_W_11, on the northeast corner is responsible for detecting the potential deadlock by checking availability of its succeeding router r_E_01. To

avoid deadlock, it informs its preceding router r_S_11 to drop the packet when router r_E_01 is not available. The LNT descriptions of arb_W_11 and r_S_11 are shown in Figure 3. The three gates "PEr_Wa_11", "Sr_Wa_11", and "n11_n01" (between square brackets) of the "arbiter_nack" process correspond to the three links (r_PE_11 → arb_W_11), (r_S_11 → arb_W_11), and (arb_W_11 → r_E_01) in Figure 2, respectively. The contents of each flit is represented by a natural number, and the arbiter process uses the variable "one_flit" of type Nat to store the flit travelling through it. The behavior of this process is a non-terminating loop with two nested choices (select[6]). The outer choice decides whether the arbiter is ready to receive a packet[7]: if its preceding router r_E_01 confirms its availability by synchronizing on the "n11_n01" gate with the arbiter, then it starts to receive the packet; or the arbiter issues a negative acknowledgement "Sr_Wa_11(false)" to r_S_11 indicating that its output is blocked by another packet. If both options are available, one is chosen nondeterministically. When the arbiter is ready to receive a packet, it nondeterministically chooses between the PE router, r_PE_11, and the south router, r_S_01. Since taking a packet from r_S_01 effectively makes an illegal turn, this arbiter first sends an acknowledgement "Sr_Wa_11(true)" to r_S_11. The router r_S_11 (represented by the LNT process "router_drop_pkt") checks the received status of arb_W_11, and a false status leads to a packet drop: r_S_11 is ready to receive the next packet which overwrites the current one that needs to be dropped.

The generated state space for the counterclockwise routing model contains terminal states, indicating deadlocks in the model. Analysis of the diagnostic sequences of transitions reveals that all four packets can get dropped by the r_S_11 router, which is an unexpected behavior. According to the routing protocol, r_S_11 should drop a packet when arb_W_11 returns a false status, and the arbiter should do so only when its output, r_E_01, is busy serving other packets. As mentioned previously, it is possible that one packet is dropped for this reason, but the remaining three should stay in the network as the network is not congested anymore. Analysis of the outer choice on the arbiter's specification shows that there always exists a path where it sends a negative acknowledgement to tell the router r_S_11 to drop its packet. The nondeterministic choice enables sending both, a true and a false acknowledgement to the router, and as long as the gate rendezvous for sending a false acknowledgement is possible, it gets a chance to occur. Therefore, arb_W_11 can always send a false acknowledgment regardless of potential deadlocks.

One possible improvement is to have a prioritized choice: the option of sending a positive acknowledgement is always the preferred one. Ideally, availability of the preferred positive option should prevent the option of sending the negative acknowledgement. To implement this priority would require that the "select" operator could probe the possibility of a gate rendezvous on the preferred choice. Implementing the priority choice in LNT requires additional processes [18], which

[6] The LNT construction "select A [] B end select" is a non-deterministic choice between A and B.

[7] In LNT, comments start with "--" and extend to the end of the line.

```
process arbiter_nack [PEr_Wa_11, Sr_Wa_11, n11_n01 : any] is
var one_flit : Nat in loop
  select
     n11_n01; -- Router r_E_01 is ready to accept packet
     select
        PEr_Wa_11(?one_flit); -- Receive packet from r_PE_11
        n11_n01(one_flit) -- Send packet to r_E_01
     [] Sr_Wa_11(true); -- Send positive ack. to r_S_11
        Sr_Wa_11(?one_flit); -- Receive packet from r_S_11
        n11_n01(one_flit) -- Send packet to r_E_01
     end select
  [] Sr_Wa_11(false) -- Send negative ack. to r_S_11
  end select
end loop end var end process

process router_drop_pkt [n10_n11, Sr_Wa_11 : any] is
var status : Bool, one_flit : Nat in loop
  n10_n11; -- Ready to accept packet from arb_N_10
  n10_n11(?one_flit); -- Receive packet from arb_N_10
  Sr_Wa_11(?status); -- Request arb_W_11's status
  -- Send packet to arb_W_11 ONLY on TRUE status
  if status then Sr_Wa_11(one_flit) end if
end loop end var end process
```

Fig. 3. The LNT processes for arb_W_11 and r_S_11.

may lead to state explosion. Even if it can be verified, the packet leakage path may not get removed due to the timing of when the probes are executed. Another option is to prune the unwanted execution paths from the generated state space using the priority operator in EXP.OPEN/SVL [19]. However, since the state space of the entire model is generated compositionally using branching bisimulation, which is not a congruence for the priority operator [20].

4.2 Removing Arbiter's Buffering Ability

After all components on the two-by-two mesh are built and connected as shown in Figure 1, the state space generation quickly becomes a challenge due to the state explosion problem. After only 10 of the 24 LNT processes are composed during verification, the state space already has reached 679,284 states, and then adding one more process leads to a state space in excess of 3 million states. Clearly, this state space growth is unmanageable. As mentioned previously, this new architecture allows multiple packets to go through one node at the same time, and the interleavings of gate rendezvous among different routing nodes is a major contributor to the exponential increase of state count. Moreover, in most cases, gate rendezvous happens with offers, which are the concrete data packets. Since there are four different packet data values, each representing the

destination location information as described in Section 4.1, interleaving of gate rendezvous is a significant source of this state explosion.

One improvement investigated to alleviate the state explosion problem is to reduce gate rendezvous between arbiters and routers in each node. This means that on the LNT model level, routers and arbiters in one node are merged into one process, removing the need for gate rendezvous between them. The resultant northwest routing node has the following behavior. It nondeterministically selects one among the following three operations: generating its own packet, receiving a packet from the northeast node, or receiving one from the southeast node. Once the node has a packet, based on the packet's destination, it attempts to send out the packet to the first choice of route, and tries alternative routes if the first one is not available. All the other three nodes have a similar behavior.

This simplification of the routing nodes indeed helps to reduce the state space. However, it removes the buffering capacity in each arbiter, and consequently causes deadlocks. A typical deadlock scenario is that initially the four routing nodes generate their own packets at the same time, between the northwest and southwest nodes, and between the northeast and southeast nodes, the packet in one node tries to go the other. No nodes can make progress in this situation. To send a packet, a node needs its neighboring node to communicate on the same gate. It is required because the node's own arbiter, which connects to the neighboring node, cannot store anything, and only the neighboring node has the storing capacity. However, if the neighboring node is trying to do the same thing to this node in the mean time, neither one can succeed in delivering packets because both are waiting for the other to accept their own packets. Removal of the arbiter's buffering ability also renders it impossible for one node to have multiple packets passing through it at the same time.

From this experiment, we conclude that arbiters in the network need storage capacity in order to relay a packet, freeing up their corresponding routing nodes to handle other communications. It also implies that simplifications on the node architecture without modifying the routing algorithm can introduce deadlocks in the system behavior. Therefore, removing interleavings of gate rendezvous on the LNT models is unsuccessful.

4.3 Finding Proper Data Abstractions

As mentioned earlier, another major contributor to the large state space is the existence of many data values in the model. The previous experiments do not consider data abstraction of a packet's content, because a router requires the packet's destination to decide its next forwarding direction: it is impossible for a router to perform routing computation without the destination information, although this information is only needed by the routers. In theory, all except the PE routers can receive packets destined to all node locations in a mesh. Since our link-fault tolerant routing algorithm allows illegal turns (c.f. Section 3), it means that a router may potentially direct packets to all of its viable directions. The idea is, thus, to abstract the routing algorithm using nondeterministic choice. In other words, after receiving a packet, a router nondeterministically selects

either its own node, indicating that the packet has reached its destination, or one of the (many) forwarding directions for the packet, without the need to examine the packet's destination. This abstraction enables us to eliminate a packet's destination information. Moreover, since every packet is provided with a preferred route and at least one alternative route for the purpose of fault-tolerance, the router's model should provide, in the non-deterministic choice, the possibility for every forwarding direction as the preferred route for a randomly destined packet, assuming the router does not perform an illegal turn. From the analysis of the routing algorithm, it is obvious that making an illegal turn is never a preferred choice for a route unless all forwarding routes of a router are illegal. For example, the north-to-east legal turn is always a preferred choice over the illegal north-to-south turn at router r_S_01 in Figure 1.

In a two-by-two mesh, there are three types of routers. First, there are routers r_S_11 and r_W_11 that can make two illegal turns (RI2). Next, there are routers r_W_10 and r_S_01 which can make one illegal turn (RI1). Finally, there are all other routers which never make illegal turns. Since routers in each of the three categories have the same abstract behavior, the rest of this section uses representatives, i.e., RI2, RI1, and RI0, to refer to routers in each category.

The next question is whether packets need to be modeled at all. Our first experiment shows that the model without packet information exhibits the packet leakage problem. As discussed in Section 4.1, the reason is an intrinsic feature of RI2, which has a nondeterministic choice where to send a packet: either to RI2 itself or to an illegal forwarding direction. Without any packet information, taking an illegal forwarding direction is always possible, regardless of deadlock avoidance, effectively creating a leakage path. To fix this problem, an abstraction of a packet has to be included such that an illegal turn in RI2 is not always possible. An important feature of the routing pattern is that a packet takes an illegal turn only after its attempt to the preferred route fails due to a failure on the route. In other words, when a packet makes an illegal turn, it must have been diverted at least once before. Thus, a packet can be modeled as a single-bit Boolean variable, indicating whether the packet has been diverted or not. In the LNT process "`router_two_illegal`" modeling RI2 shown in Figure 4, only a diverted packet can take illegal turns. This restriction rules out the possibility of dropping packets which have not taken an alternate route yet.

Comparing the precise routing decision computation with the nondeterministic choice in the abstract model, a packet destined to one forwarding direction in the concrete model has the possibility to be forwarded to any routing direction in the abstract model. Therefore the abstraction is conservative in that it preserves all transition sequences in its corresponding concrete model. One subtle difference introduced in the abstract model is the notion of a diverted packet, which does not exist in the concrete model. It is, however, a feature that implicitly exists in the concrete model's routing behavior.

There are also three categories of arbiters, corresponding to the router categories. Figure 5 shows the arbiter corresponding to RI2. It selects between its PE router and two routers, flits from which may just have made an illegal turn.

```
process router_two_illegal [input, out_arb_PE, out1_illegal,
                            out2_illegal, drop : any] is
var one_flit, arb_status : Bool in loop
  input(?one_flit);
  select
     out_arb_PE(one_flit)
  []
     if one_flit == true then -- packet is diverted
       -- first try out1_illegal, then out2_illegal
       out1_illegal(?arb_status);
       if arb_status == true then
         out1_illegal(one_flit)
       else
         out2_illegal(?arb_status);
         if arb_status == true then
           out2_illegal(one_flit)
         else
           drop -- both illegal turns impossible
         end if
       end if
     end if
  end select
end loop end var end process
```

Fig. 4. The LNT process for the RI2 router.

For each option the arbiter takes, after receiving a flit, it keeps rejecting requests from RI2 routers until it delivers the flit. When receiving rejections on all its illegal forwarding routes, RI2 drops the packet to prevent potential deadlock. The complete LNT specification for the two-by-two NoC is available at `http://www.async.ece.utah.edu/~zhangz/research/lnt_modeling/`.

5 Verification Results

Verification of the NoC model consists of two steps: generating the *Labeled Transition System* (LTS) from the LNT specification, and then analyzing the LTS to verify properties of interest. The LTS for each investigated model is generated compositionally, i.e., by generating and minimizing the LTSs for each process separately before combining them to the LTS of the complete system. For the combination steps, our verification applied smart reduction [21], which uses heuristics to find an optimal ordering of composition and minimization steps to keep the intermediate state spaces manageable. Minimization is performed with respect to divergence-sensitive branching bisimulation equivalence [22]. Thus, any livelocks found are preserved during composition, whereas using standard branching bisimulation would collapse every livelock into a deadlock. In other

```
process arbiter_nack_2 [in_PE_router, in1_illegal,
                        in2_illegal, arb_out : any] is
var one_flit : Bool in loop
  select
     in_PE_router(true); in_PE_router(?one_flit);
     loop L1 in select
        arb_out(one_flit); break L1
     [] in1_illegal(false)
     [] in2_illegal(false)
     end select end loop -- L1
  []
     in1_illegal(true); in1_illegal(?one_flit);
     loop L2 in select
        arb_out(one_flit); break L2
     [] in1_illegal(false)
     [] in2_illegal(false)
     end select end loop -- L2
  []
     in2_illegal(true); in2_illegal(?one_flit);
     loop L3 in select
        arb_out(one_flit); break L3
     [] in1_illegal(false)
     [] in2_illegal(false)
     end select end loop -- L3
  end select
end loop end var end process
```

Fig. 5. The LNT process for the arbiter corresponding to RI2.

words, divergence-sensitive branching bisimulation can reveal true deadlock scenarios. The three properties of interest are: (1) the link-fault tolerant routing algorithm is free of deadlocks; (2) given at most one failure link, it is never the case that a router is unable to route a packet; and (3) given at most one failure link, a packet never gets dropped when there is only one packet in the network.

Table 1 shows the LTS information for nine two-by-two mesh models: the first row represents a mesh without any link failure, and the remaining eight rows each represent the same NoC with one failure link whose location is shown in the first column. The columns under "Intermediate LTS" show the number of states and transitions of the largest intermediate LTS, and the columns under "Final LTS" show those of the final LTS. The two columns under "Performance" display the maximal amount of allocated virtual memory (in MB) and the total execution time (in s) to generate each LTS. A desktop machine with a CPU of eight 3.60 GHz cores and 16GB of available RAM is used to generate the results listed in this table. One core is used at any time for the parallel composition and state minimization steps. The last column shows the labels of each LTS.

Table 1. LTS's for two-by-two NoCs.

Failure Link	Interm. LTS Size States	Transitions	Final LTS St.	Tr.	Performance RAM	Time	Labels
none	6,295,773	83,386,208	1	1	32,945	5,976	i
01 → 00	20,340	193,726	41	224	111	83	i, drop_Sr_11, drop_Wr_11
01 → 11	1,369,068	18,221,153	1	3	4,039	499	i, drop_Sr_01, drop_Sr_11
00 → 10	6,560	50,688	21	104	111	80	i, drop_Sr_11, drop_Wr_11
00 → 01	6,560	50,688	21	104	111	81	i, drop_Wr_11, drop_Sr_11
10 → 11	122,724	1,269,981	1	3	111	89	i, drop_Wr_10, drop_Wr_11
10 → 00	20,340	193,726	41	224	111	80	i, drop_Wr_11, drop_Sr_11
11 → 01	367,200	4,172,652	1	3	111	106	i, drop_Sr_11, drop_Wr_11
11 → 10	367,200	4,172,652	1	3	111	105	i, drop_Sr_11, drop_Wr_11

The final LTS for each model is generated by hiding all gates that represent the links between the routers and the arbiters. The only two visible types of gates are the route-failure gates and the packet-drop gates. Rendezvous on the former happen when a router has exhausted all options to forward a packet; rendezvous on the latter occur when a router drops a packet. These gates are left visible to facilitate the verification tasks (2) and (3) described above. To model a single link fault in LNT, a working arbiter process is replaced by an arbiter that sends false status to all its connected input routers.

The final LTSs show similarities between the following pairs: (01 → 00, 10 → 00),(01 → 11, 10 → 11), (00 → 10, 00 → 01), (11 → 01, 11 → 10). All visible labels on these LTSs are packet-drop labels. Based on the three types of the router introduced previously, renaming these labels to "drop-at-RI2", and "drop-at-RI1" produces bisimilarity between the two LTSs in each pair.

Since deadlock freedom is a global property and is not always possible to be inferred from local LTS states, generating the global LTS is necessary. A system has a deadlock if its LTS contains a state/transition sequence that starts from the initial state and ends in a terminal state, i.e., a state without outgoing transitions. Deadlock detection then becomes a search for such states in the LTS. Using the CADP toolbox, it is found that no such sequence exists in any NoC's LTS in Table 1. Since the entries in this table cover all possible one-link fault configurations, we can conclude that the link-fault tolerant algorithm is free of deadlock for a two-by-two mesh.

To prove that a router is always able to route a packet, it is necessary to verify that no route-failure gate rendezvous occurs. Since these gates are not hidden during parallel composition, it is straightforward to check their existence in each LTS. Table 1 shows that no transitions are labeled with route-failure labels, which proves verification task (2).

The transition labels in Table 1 show that with one failure link in the network, packets may be dropped, namely when the attempt to make an illegal turn potentially could cause a deadlock. Therefore, in a highly congested network, dropping packets is likely to happen. On the other hand, the routing algorithm should not drop any packets if making an illegal turn is safe. One simplification

for checking this property is to have only one node generate only one packet, and verify that no packet-drop labels exist on each model's LTS. For each link failure location shown in Table 1, the packet can be generated in any of the four nodes. Therefore, this property is thoroughly checked in all 36 possible models. No drop-packet label is found in all LTS's, which proves that no packet is dropped when there is only one packet in the network.

6 Discussion

The construction and refinement of the two-by-two NoC model taught us several valuable lessons. The counterclockwise routing example reveals a packet leakage path in the arbiters that instructs their preceding routers to drop the packet. This leakage stems from the arbiter's specification, in that each arbiter must check its succeeding router's availability before it can receive a packet from another router. For example, arb_W_11 must check with r_E_01 before it receives a packet from either r_PE_11 or r_S_11. Otherwise, if the arbiter does not receive its succeeding router's acknowledgement, it sends a "drop" signal to its preceding router. This option is modeled simply as the arbiter sending back the "drop" signal, which opens the path for packet leakage. The second lesson is that it is necessary for an arbiter to have a buffering capacity for the proposed routing architecture because an arbiter does not need to guarantee the availability of its succeeding router before it receives a packet. It is this idea that leads us to redesign the arbiters, such as the one shown in Figure 5. Without formal analysis, it would have been very challenging to discover the diagnostic information that shows these flaws in our arbiter design.

The state explosion problem encountered during the evolutions of our NoC models inspired us to come up with an adequate data abstraction. This process provided us with a deeper understanding of the routing algorithm. The resultant changes on the router and arbiter models show interesting symmetries that we thought did not exist before. Previous attempts to find symmetries between two nodes did not succeed due to the mismatch in terms of illegal turns made in different nodes. With the data abstraction, routers can be categorized into RI0, RI1, and RI2, as described previously, and each category corresponds to one type of arbiter, as well. The relative positions and connections between these routers and arbiters in Figure 1 show strong symmetries between the clockwise and counterclockwise cycles. Experiences gained in this process may help us to develop heuristics to automate the search for symmetries in the LNT model description so that the model checking effort can be reduced by focussing on representatives from each symmetry group. Efficient state reduction techniques can potentially allow us to perform model checking on larger-scale networks.

With the data abstraction presented in this paper, our results verify deadlock freedom and one-link-fault tolerance of the proposed routing algorithm, demonstrating its robustness. As for its efficiency, our results prove that a packet never gets dropped when it is the only one in the network. Our preliminary experiments show that even when one node generates only two packets, it is possible

that the first packet occupies the output link of the second, which attempts to make an illegal turn and gets dropped due to deadlock avoidance. Therefore, it is a challenging task to justify the performance of the routing algorithm in terms of packet drop rate due to deadlock avoidance in the current setting. Since it is directly related to the link failure probability, performance evaluation requires annotations of link failure probability in the model and stochastic model checking techniques are needed to provide a quantitative measure of its performance. Another challenge is that with the current data abstraction, proving delivery of every packet is difficult, since it is possible that some, if not all, packets produced get continuously dropped in the network. A simple solution would be to allocate a unique identifier to each packet and check them on both the production and absorption ends, but the resultant state space is likely to become unmanageable. A more suitable data abstraction scheme and more advanced state reduction techniques, such as on-the-fly model checking [23], are needed to meet this challenge.

References

1. Vivet, P., Lattard, D., Clermidy, F., Beigne, E., Bernard, C., Durand, Y., Durupt, J., Varreau, D.: Faust, an asynchronous network-on-chip based architecture for telecom applications. Proc. 2007 Design, Automation and Test in Europe (DATE07) (2007)
2. Hoskote, Y., Vangal, S., Singh, A., Borkar, N., Borkar, S.: A 5-ghz mesh interconnect for a teraflops processor. Micro, IEEE **27**(5) (Sept 2007) 51–61
3. Wu, J., Zhang, Z., Myers, C.: A fault-tolerant routing algorithm for a network-on-chip using a link fault model. In: Virtual Worldwide Forum for PhD Researchers in Electronic Design Automation. (2011)
4. Fick, D., DeOrio, A., Chen, G., Bertacco, V., Sylvester, D., Blaauw, D.: A Highly Resilient Routing Algorithm for Fault-tolerant NoCs. In: Proceedings of the Conference on Design, Automation and Test in Europe, European Design and Automation Association (2009) 21–26
5. Hosseini, A., Ragheb, T., Massoud, Y.: A fault-aware dynamic routing algorithm for on-chip networks. In: ISCAS, IEEE (2008) 2653–2656
6. Glass, C.J., Ni, L.M.: Fault-tolerant wormhole routing in meshes. In: FTCS, IEEE Computer Society (1993) 240–249
7. Imai, M., Yoneda, T.: Improving dependability and performance of fully asynchronous on-chip networks. In: Proceedings of the 2011 17th IEEE International Symposium on Asynchronous Circuits and Systems. ASYNC '11, Washington, DC, USA, IEEE Computer Society (2011) 65–76
8. Borrione, D., Helmy, A., Pierre, L., Schmaltz, J.: A formal approach to the verification of networks on chip. EURASIP J. Embedded Syst. **2009** (January 2009) 2:1–2:14
9. Helmy, A., Pierre, L., Jantsch, A.: Theorem proving techniques for the formal verification of NoC communications with non-minimal adaptive routing. In: DDECS, IEEE (2010) 221–224
10. Borrione, D., Boubekeur, M., Mounier, L., Renaudin, M., Sirianni, A.: Validation of asynchronous circuit specifications using IF/CADP. In: IFIP Intl. Conference on VLSI, Darmstadt, Germany. (December 2003)

11. Salaün, G., Serwe, W.: Translating Hardware Process Algebras into Standard Process Algebras : Illustration with CHP and LOTOS. Technical Report RR-5666, INRIA (September 2005)
12. Salaün, G., Serwe, W., Thonnart, Y., Vivet, P.: Formal verification of CHP specifications with CADP illustration on an asynchronous Network-on-Chip. In: Asynchronous Circuits and Systems, 2007. ASYNC 2007. 13th IEEE International Symposium on. (March 2007) 73–82
13. Beigné, E., Clermidy, F., Vivet, P., Clouard, A., Renaudin, M.: An Asynchronous NOC Architecture Providing Low Latency Service and Its Multi-Level Design Framework. In: ASYNC, IEEE Computer Society (2005) 54–63
14. Chiu, G.M.: The odd-even turn model for adaptive routing. IEEE Trans. Parallel Distrib. Syst. **11**(7) (July 2000) 729–738
15. Myers, C.J.: Asynchronous circuit design. Wiley (2001)
16. Champelovier, D., Clerc, X., Garavel, H., Guerte, Y., McKinty, C., Powazny, V., Lang, F., Serwe, W., Smeding, G.: Reference manual of the LNT to LOTOS translator (version 6.0). INRIA/VASY/CONVECS. (June 2014)
17. Garavel, H., Lang, F., Mateescu, R., Serwe, W.: CADP 2011: a toolbox for the construction and analysis of distributed processes. STTT **15**(2) (2013) 89–107
18. Garavel, H., Salaün, G., Serwe, W.: On the Semantics of Communicating Hardware Processes and their Translation into LOTOS for the Verification of Asynchronous Circuits with CADP. Science of Computer Programming (2009)
19. Garavel, H., Lang, F.: SVL: a Scripting Language for Compositional Verification. In: Proceedings of the 21st IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems FORTE'2001, Kluwer Academic Publishers (August 2001) 377–392 Full version available as INRIA Research Report RR-4223.
20. Gazda, M., Fokkink, W.: Congruence from the operator's point of view: Compositionality requirements on process semantics. In: SOS. Volume 32 of EPTCS. (2010) 15–25
21. Crouzen, P., Lang, F.: Smart Reduction. In: Proceedings of Fundamental Approaches to Software Engineering (FASE'2011). Volume 6603 of Lecture Notes in Computer Science., Saarbrucken, Allemagne, Springer Verlag (March 2011) 111–126
22. van Glabbeek, R.J., Luttik, B., Trcka, N.: Branching bisimilarity with explicit divergence. Fundam. Inform. **93**(4) (2009) 371–392
23. Mateescu, R., Thivolle, D.: A model checking language for concurrent value-passing systems. In: Proceedings of the 15th International Symposium on Formal Methods FM'08 (Turku, Finland). Volume 5014. (May 2008) 148–164