

# Theorem Proving Techniques for the Formal Verification of NoC Communications with Non-minimal Adaptive Routing

Amr HELMY, Laurence PIERRE  
TIMA Laboratory (CNRS, GrenobleINP, UJF)  
Grenoble, France  
Amr.Helmy@imag.fr, Laurence.Pierre@imag.fr

Axel JANTSCH  
Royal Institute of Technology  
Stockholm, Sweden  
axel@kth.se

**Abstract**—This paper focuses on the formal verification of communications in Networks on Chip. We describe how an enhanced version of the *GeNoC* proof methodology has been applied to the Nostrum NoC which encompasses various non-trivial features such as a defective non-minimal routing algorithm. We demonstrate how the features of the Nostrum protocol layers can be captured by the current version of *GeNoC* that enables a step-by-step formalization of communication operations while taking various protocol details into consideration. We prove that packets arrive properly and that packets are never lost. Also, we prove the soundness of the Nostrum data link, network and transport layers.

## I. INTRODUCTION

A variety of packet switched communication networks have been proposed for providing general purpose and high performance on-chip communication between multiple cores, IPs and memories. An NoC includes many different aspects on several levels of the protocol stack, from the physical to the data link, to the network and the transport layer. Verification of each function individually and of the network as a whole is hard. Traditional simulation has to be complemented by more *formal techniques*, in particular when guaranties about loss-less delivery, deadlock freeness and other Quality of Service properties are concerned.

Most proposed solutions for performance, traffic or behavior analysis for NoCs are simulation or emulation-oriented [1], [2], [3]. Few approaches address the use of formal or semi-formal methods. A methodology to use checkers for temporal assertions is proposed in [4]. It targets a two-level hierarchical ring structure, and PSL properties are used to express interface-level requirements. [5] advocates communication-centric debug instead of computation-centric debug for complex SoC's, and also uses a monitor-based solution. Both approaches rely on a formal expression of properties (assertions), but are indeed founded on a dynamic, non formal, execution/emulation of a detailed design, where all architectural parameters and implementation decisions have been settled. In contrast, the work reported here is fully generic and relies on formal verification within a theorem prover. Our method for verifying the correctness of the communication operations

is based on a consequential enhancement of a *generic formal model* for the representation of the communications on a chip, *GeNoC* [6], encoded in the ACL2 logic [7]. The new version of *GeNoC* [8] makes it possible to describe the progression of messages in the network step-by-step. Communication operations are described faithfully, from the transport layer to the data link layer.

Using as example the verification of the Nostrum NoC, we demonstrate here for the first time the adaptability of this model to complex infrastructures that encompass various non-trivial features such as elaborate priority mechanisms and a defective non-minimal routing algorithm. To that goal, the last *GeNoC* version has been enriched with some additional aspects.

## II. VERIFICATION PRINCIPLES

The formal meta-model represents the transmission of messages on a *generic* communication architecture, with an *arbitrary* network characterization (topology and node interfaces), flow control mechanism, routing algorithm and switching technique, only characterized by representative properties (proof obligations). The main function of this model is called here *GeNoC<sub>f</sub>*.

The original model [6] mainly relies on the following functions: (1) interfaces are represented by two functions, *send* to inject frames on the network, and *recv* to receive frames, (2) the routing algorithm is represented by function *Routing*, and the switching technique is represented by function *Scheduling*. The original *GeNoC<sub>f</sub>* function represents *atomically* the transfer of messages *from their source to their destination*. It expresses how a message travels in the network, but the granularity is not the time steps of this movement.

Our current version results in new modules that contain an explicit representation of the global state of the network, and explicit time stamps for the emission of messages<sup>1</sup>. Function

<sup>1</sup>In addition to its data, a *message* contains: its source, destination, current position, the time at which it is injected in the network, and optionally some additional data such as its number of flits or its number of hops. It is also uniquely identified by a natural number *id*.

*R4D* (“ready for departure”) determines which messages can be in the network at the current time. Functions *Routing* and *Scheduling* now enable a *step-by-step simulation of the evolution of the network state*. At the Data Link layer, the transmission protocol can be specified.

The main function  $GeNoC_f$  takes as parameters: a list of messages emitted at source nodes ( $\mathbf{M}$ ), the set of addresses of the network nodes ( $\mathbf{a}$ ), a *finite* number of attempts ( $\mathbf{att}$ ), an accumulator for the set of arrived messages ( $\mathbf{AM}$ , which is originally empty), the current state of the network ( $\mathbf{st}$ ), and the current time ( $\mathbf{t}$ ).

**Correctness theorems** are associated with the  $GeNoC$  model. Roughly speaking, property (1) states that for all topology  $\mathcal{T}$ , access control  $\mathcal{C}$ , routing algorithm  $\mathcal{R}$ , and scheduling policy  $\mathcal{S}$  that satisfy given constraints  $P_1, P_2, P_3$ , and  $P_4$ ,  $GeNoC$  fulfills a correctness property  $\wp$  that asserts that *every message arrived at some node  $n$  was actually issued at some source node  $s$  and originally addressed to node  $n$ , and that it reaches its destination without modification of its content*.

$$\forall \mathcal{T} \forall \mathcal{C} \forall \mathcal{R} \forall \mathcal{S}, P_1(\mathcal{T}) \wedge P_2(\mathcal{C}) \wedge P_3(\mathcal{R}) \wedge P_4(\mathcal{S}) \\ \Rightarrow \wp(GeNoC(\mathcal{T}, \mathcal{C}, \mathcal{R}, \mathcal{S})) \quad (1)$$

Constraints (or proof obligations)  $P_1, P_2, P_3$ , and  $P_4$ , described in [8], express essential properties of the key components.  $\wp(GeNoC(\mathcal{T}, \mathcal{C}, \mathcal{R}, \mathcal{S}))$  is a logical consequence of these constraints, and the proof of (1) is performed once and for all, *without considering the actual definitions of the constituents*.

We have recently improved the model with another property that states that, under constraints  $P'_1, P'_2, P'_3$ , and  $P'_4$ ,  $GeNoC$  fulfills a property  $\wp_2$  that asserts that *no message is lost* (for instance by the transmission protocol). Constraints  $P'_1, P'_2, P'_3$ , and  $P'_4$  are close to  $P_1, P_2, P_3$ , and  $P_4$ , plus some additional obligations. Some of them will be sketched in table I, section V.

A more precise statement of properties  $\wp$  and  $\wp_2$ , using function  $GeNoC_f$  is as follows.

*Theorem 1.* For each arrived message (*i.e.*, each element of the returned  $AM$  denoted  $GeNoC_f.AM(M, a, att, AM, st, t)$ ), there exists a message in  $M$  with the same *id*, the same destination (*dest*) and the same content, denoted *frm*:

$$\forall m \in GeNoC_f(M, a, att, AM, st, t).AM, \exists m' \in M, \\ \left\{ \begin{array}{l} m.id = m'.id \wedge m.dest = m'.dest \\ \wedge m.frm = m'.frm \end{array} \right.$$

*Theorem 2.* *No message is lost* *i.e.*, the union of the identifiers of the arrived and en route messages is equal to the original set of message identifiers:

$$(GeNoC_f(M, a, att, AM, st, t).AM_{id} \\ \cup GeNoC_f(M, a, att, AM, st, t).M_{id}) = M_{id}$$

The global correctness of the network model is preserved for all particular definitions satisfying the constraints. Thus,

for any **new instance of network**, *i.e.*, for any  $\mathcal{T}_0, \mathcal{C}_0, \mathcal{R}_0$  and  $\mathcal{S}_0$  actually defined, it is sufficient to discharge proof obligations  $P_1(\mathcal{T}_0), P_2(\mathcal{C}_0), P_3(\mathcal{R}_0)$ , and  $P_4(\mathcal{S}_0)$ , since  $\wp(GeNoC(\mathcal{T}_0, \mathcal{C}_0, \mathcal{R}_0, \mathcal{S}_0))$  is a logical consequence of these properties (and similarly for  $\wp_2$ ). Hence, when creating a new NoC instance, the user simply has to:

- define all the functions that model  $\mathcal{T}_0, \mathcal{C}_0, \mathcal{R}_0$  and  $\mathcal{S}_0$ , such as functions *R4D, Routing* and *Scheduling*,
- discharge (verify) the associated proof obligations.

### III. OVERVIEW OF NOSTRUM

Nostrum [9] is a non-minimal adaptive routing network. Its topology is a 2D-mesh. Packets are never buffered or blocked in a switch, a packet is either ejected from the network to its destination node or it is forwarded to a neighboring switch. When two packets contend for the same output link, one wins and the other is deflected to another output link. Thus, the technique is called *deflective routing* or *hot-potato routing*, because packets are never kept for more than one switching cycle [10]. This leads to small switches with no buffers except at the input and output ports.

Nostrum has a number of other features, note relevant in this context. The routing procedure is most interesting and explained in more detail in the following: The packets at the five input ports are inspected. All packets include the relative target address in their header. The target address is a tuple  $(x, y)$ , with  $-k < x, y < k$ , where  $k$  is the number of nodes in each dimension of the mesh. The table below shows the possible output directions.

$x$	$y$	Output direction
0	0	Local port
0	< 0	South
0	> 0	North
< 0	0	West
< 0	< 0	South-West
< 0	> 0	North-West
> 0	0	East
> 0	< 0	South-East
> 0	> 0	North-East

First, the router determines the desirable directions of a packet. Then, packets are ordered according to a priority function. Packets pick their favored output port in the order determined by their priority. Packets that lose the competition for a desirable output direction, are deflected to an undesirable output port. Three decisions have to be taken in this procedure: priority of packets, selection of desirable and of undesirable output ports.

#### A. Priority function

The priority function uses the hop count of a packet and its distance to the destination. Higher hop count gives higher priority. If the hop count is the same, packets that are closer to their destination obtain higher priority. If two packets have the same priority according to both criteria, it is undefined which gets preference. If older packets (*i.e.* higher hop count) have priority over younger packets, livelocks can be avoided.

### B. Selection of a desirable output port

If a packet has the luxury that both desirable output ports are free, then the direction is preferred in which the packet has to travel longer. E.g. if  $x = -3$  and  $y = +2$ , the West port is picked, because  $|x| > |y|$ . If there is the same distance in both directions it is undefined which output direction is chosen.

This technique preserves more freedom for a packet, which may be useful later on the route. In general we observe that the more freedom packets have in choosing their direction, the fewer deflections occur. Hence, preserving freedom of choice is an effective way to decrease the number of deflections.

### C. Selection of an undesirable output port

If a packet loses the competition for a desirable output direction, a non-desirable port has to be chosen. In principle, all ports are equally bad in that the distance to the destination is increased by two, no matter which undesirable direction is picked. Nostrum uses two criteria to select the deflection port. First, stress values of the neighbors are considered. Each switch communicates its own load to all its neighbors through special control wires. They indicate the level of load a switch experiences. Undesirable ports are chosen such that packets are deflected to switches which experience a lower load. The second criterion used in Nostrum is the direction of the network center. Under many traffic patterns the center of the network is a highly congested area, because for many packets the way through the center is shortest. Thus, deflecting packets away from the center can help to relax the load in the center. Therefore, in Nostrum there is a preferred order of deflection ports which depends on the location in the network. E.g. in the North-Western part of the network, packets are preferably deflected to the West and the North, rather than the South and the East.

## IV. GeNoC INSTANCE - THE NOSTRUM CASE

We sketch the Nostrum instantiation, according to the principles described in section II, with an emphasis on the definition of the *Routing* function.

### A. Routing function

With the deflective routing strategy of Nostrum, computing all possible routes from the current position to the destination is clearly impossible. In its initial versions [6], [11], the *GeNoC* model was only able to consider *atomically* the moves of messages from their sources to their destinations, hence the concept of *route* was crucial and undividable. It was impossible to deal with non-minimal adaptive routing algorithms. The new version of *GeNoC* allows to reason on the *step-by-step* progression of messages in the network and offers more flexibility. The notion of route is still present (and convenient in the case of deterministic algorithms), but only the next step node is actually significant for non-deterministic ones. Thus a simple solution to model the Nostrum routing is as follows. At each step, only 4 possible routes are computed (for messages that have not reached their destination node): the next step node of each route corresponds to one of the

four neighbours, and the rest of the route is computed by a function *NostrumRoutingLogic* that recursively calls the following function *NostrumRLogic* to complete the route to the destination. This sub-route is updated in the next recursive call, hence preserving its accuracy. Function *NostrumRLogic* implements the policy given in section III.

```
function NostrumRLogic(curr, dest) {
if |dest.y - curr.y| > |dest.x - curr.x| then
  if dest.y < curr.y then return move-north(curr)
  else return move-south(curr)
else
  if curr.x < dest.x then return move-east(curr)
  else return move-west(curr) }
```

Function *NostrumRCore* computes the list of all possible routes for one message. To respect the policy of section III-B, this list of routes is sorted according to the preference of the routes: if possible the message should get closer to its destination while taking the direction in which it has to travel longer. Eight auxiliary functions are used. They play similar roles, for instance *north\_i\_hop\_route* computes the route such that the next step node is the north neighbour and the rest of the route is computed according to function *NostrumRoutingLogic*.

```
function NostrumRCore (c,d) {
if c.dir = i then // message is on an input port
  { nr=north_i_hop_route(c,d);
    sr=south_i_hop_route(c,d);
    wr=west_i_hop_route(c,d);
    er=east_i_hop_route(c,d);
    if c.x = d.x & c.y = d.y then
      list(local_route(c), nr, er, sr, wr);
    else
      if |dest.y - curr.y| > |dest.x - curr.x| then
        if d.y < c.y then
          if d.x < c.x then list(nr, wr, sr, er);
          else list(nr, er, sr, wr);
        else
          if d.x < c.x then list(sr, wr, nr, er);
          else list(sr, er, nr, wr);
      else
        if d.x < c.x then
          if d.y < c.y then list(wr, nr, sr, er);
          else list(wr, sr, nr, er);
        else
          if d.y < c.y then list(er, nr, sr, wr);
          else list(er, sr, nr, wr);
    }
  else // message is on output port, it crosses the link
    { switch (c.port)
      case N: return north_o_hop_route(c,d);
      case S: return south_o_hop_route(c,d);
      case E: return east_o_hop_route(c,d);
      case W: return west_o_hop_route(c,d); }
}
```

OSI layer	Main proof obligations	Total CPU time
Transport layer	The composition of the encoding and decoding functions is the identity. The intersection of lists <b>TR</b> and <b>Delayed</b> produced by <i>R4D</i> is empty. The union of lists <b>TR</b> and <b>Delayed</b> produced by <i>R4D</i> equals <b>M</b> .	9.7 s
Network layer	<b>Routing:</b> Function <i>Routing</i> terminates (recursion is well-founded). Each route from a current node <i>n</i> to a destination <i>d</i> actually starts in <i>n</i> , uses only valid nodes, and ends in <i>d</i> . <b>Scheduling:</b> The lists of arrived and <i>en route</i> messages ( <b>Arr</b> and <b>TM</b> ) are well-formed. The intersection of these two lists of messages <b>Arr</b> and <b>TM</b> is empty. The union of these two lists equals list <b>TR</b> produced by <i>R4D</i> .	524.7 s 108.5 s
Data link layer	The network state obtained after transmitting the messages (one hop) is a valid state.	4.7 s

TABLE I  
MAIN CONSTRAINTS OF THE NOC CONSTITUENTS

Finally, function *Routing* is instantiated by function *NostrumRouting* that uses *NostrumRCore* to compute the routes for all the messages.

### B. Scheduling function

Nostrum uses a packet switching technique. One of the advantages of *GeNoC* is that, once the core packet switching technique (or any other constituent) has been formalized and verified for a given NoC, it can straightforwardly be adapted to any other network. This is the case here, we reuse the packet scheduling core function defined for the Spidergon NoC in [12]. This function, called *NostrumPS* here, mainly returns the list of messages that are still *en route*, and those that have reached their destination.

Function *NostrumPScheduling* is the instance of function *Scheduling*. It calls function *NostrumPS* to process the messages, previously sorted according to their age (hop count), in decreasing order. If two messages have the same hop count, the priority is given to the message that is closer to its destination (section III-A).

Finally, we recall that each node keeps track of a history of its load during the previous cycles, to be able to optimize the choice between possible routes in case one of the desirable routes cannot be taken (section III-C): the actual route can be chosen in the direction of the less loaded neighbours. Function *NostrumTestRoutes*, used by *NostrumPS*, returns the route that will likely be taken according to this policy (considering the average loads of the neighbouring nodes), or no route if no move is feasible (which is possible if the message is trying to leave a local port towards the network).

## V. RESULTS AND CONCLUSIONS

Table I outlines the main proof obligations for all the key constituents. A total number of 68 functions were defined in ACL2 to instantiate all the constituents of the model, and 272 theorems (proof obligations + some auxiliary theorems) have been proven, for a total CPU time of about 12 minutes on an Intel Core Duo, 1.6 GHz.

NoCs are complex communication systems realizing a sophisticated protocol stack in a distributed and concurrent implementation. It is non-trivial to ensure their correctness. With the extended *GeNoC* model, we have analyzed some

essential properties of Nostrum, a state of the art NoC. We have shown that packets arrive properly and that packets are never lost. Since the method in [13] allows for calculating the maximum delay, we could prove the soundness of the Nostrum data link, network and transport layers.

## REFERENCES

- [1] L. Ost, A. Mello, J. Palma, F. Moraes, and N. Calazans, "MAIA - a framework for networks on chip generation and verification," in *Proc. ASP-DAC*, 2005.
- [2] K. Goossens, J. Dielissen, O. Gangwal, S. Pestana, A. Radulescu, and E. Rijpkema, "A Design Flow for Application-Specific Networks on Chip with Guaranteed Performance to Accelerate SOC Design and Verification," in *Proc. DATE'05*, 2005.
- [3] N. Genko, D. Atienza, and G. D. Micheli, "NoC Emulation on FPGA: HW/SW Synergy for NoC Features Exploration," in *Proc. ParCo'05*, 2005.
- [4] J. Chenard, S. Bourduas, N. Azuelos, M. Boulé, and Z. Zilic, "Hardware Assertion Checkers in On-line Detection of Network-on-Chip Faults," in *Proc. Workshop on Diagnostic Services in Networks-on-Chips*, 2007.
- [5] K. Goossens, B. Vermeulen, R. van Steeden, and M. Bennebroek, "Transaction-Based Communication-Centric Debug," in *Proc. International Symposium on Networks-on-Chip*, 2007.
- [6] J. Schmaltz and D. Borrione, "A functional formalization of on chip communications," *Formal Aspects of Computing*, vol. 20, no. 3, May 2008.
- [7] M. Kaufmann, P. Manolios, and J. Moore, *Computer Aided Reasoning: an Approach*. Kluwer, 2002.
- [8] D. Borrione, A. Helmy, L. Pierre, and J. Schmaltz, "A Formal Approach to the Verification of Networks on Chip," *EURASIP Journal on Embedded Systems*, 2009.
- [9] M. Millberg, E. Nilsson, R. Thid, S. Kumar, and A. Jantsch, "The Nostrum backbone - a communication protocol stack for networks on chip," in *Proc. of the VLSI Design Conference*, Mumbai, India, January 2004.
- [10] E. Nilsson, M. Millberg, J. Öberg, and A. Jantsch, "Load distribution with the proximity congestion awareness in a network on chip," in *Proceedings of DATE'03*, March 2003.
- [11] D. Borrione, A. Helmy, L. Pierre, and J. Schmaltz, "A Generic Model for Formally Verifying NoC Communication Architectures: A Case Study," in *Proc. IEEE International Conference on NoCs (NoCs'07)*, May 2007.
- [12] —, *Networks-on-Chips: Theory and Practice*. CRC Press (Taylor and Francis group), 2009, ch. "Formal Verification of Communications in Networks-on-Chip".
- [13] A. Jantsch, "Models of computation for networks on chip," in *ACSD '06: Proc. of the Sixth International Conference on Application of Concurrency to System Design*. Washington, DC: IEEE Computer Society, 2006.

<sup>1</sup>This work is partly supported by the French project SoCKET (FCE)