Formal Specification of Networks-on-Chips: Deadlock and Evacuation

Freek Verbeek and Julien Schmaltz
Institute for Computing and Information Sciences, Radboud University Nijmegen
School of Computer Science, Open University of The Netherlands
E-mail: f.verbeek@cs.ru.nl, julien.schmaltz@ou.nl

***Abstract*—Networks-on-chips (NoC) are emerging as a promising interconnect solution for efficient Multi-Processors Systems-on-Chips. We propose a methodology that supports the specification of parametric NoCs. We provide sufficient constraints that ensure deadlock-free routing, functional correctness, and liveness of the design. To illustrate our method, we discharge these constraints for a parametric NoC inspired by the HERMES architecture.**

I. Introduction

Multi-Processors Systems-on-Chip (MPSoC) designs integrate several processing and memory cores on a single die. To gain performance the trend is to increase the number of cores and parallelism [1]. To manage the complexity of modern MPSoCs, a popular approach is *platform based design* [2]. A new MPSoC is built by assembling pre-designed components according to a generic architecture. Complexity is also reduced by raising the abstraction level of the initial design phase. Abstractions and the interconnect become crucial. With the increase of the number of interconnected cores, buses no longer offer an adequate solution. On-chip complex networks are required [3], [4]. As for processing elements, the formal guarantee of their correct behavior will become mandatory.

NoCs are complex parametric designs making their formal analysis challenging. Regarding the communication infrastructure, Schmaltz *et al.* [5], [6] propose GeNoC as an efficient specification environment for abstract and parametric NoC designs. GeNoC provides (1) a network model to specify the main characteristics of the NoC (e.g., topology, routing); (2) an interpreter defining how the different constituents interact; (3) a correctness theorem for the interpreter; and (4) sufficient constraints – or proof obligations – on the constituents from which the proof of the correctness theorem follows. GeNoC is generic in the sense that the constituents are not given any specific definition but only characterized by their proof obligations. Consequently, the validation of a particular NoC reduces to (1) giving a concrete definition to the constituents and (2) discharging the corresponding instantiated constraints. GeNoC has been implemented in the logic of the ACL2 theorem proving system [7] and applied to several case-studies, e.g., the HERMES [8] and Spidergon [6] designs. Instances of GeNoC are *executable*. They can be simulated on concrete data. The same model is used for simulation and validation.

The correctness theorem of GeNoC states that when message m reaches destination node d message m was emitted at a valid source node, was actually destined to node d, and followed a valid path to d. This is rather weak as it trivially holds if no message ever each a destination. Our original contribution is the addition of two new theorems to GeNoC. The first one ensures that the routing function is deadlock-free. The second one proves that all messages injected in the network eventually leave the network, i.e., liveness. To prove the original GeNoC correctness theorem, one has to prove that the routing function terminates and ends in the correct destination. This only proves that computing *one route* for *one message* is correct *when there is no other message in the network*. Deadlock and evacuation are global properties that depend on the interaction of several messages. Their proof requires more than termination of one application of the routing function. This addition is the main contribution for this paper. Following the GeNoC approach, our theorems are generic and we provide sufficient proof obligations as well. We instantiate our theorems on a parametric 2D-mesh inspired from the HERMES NoC. The result is an executable specification of a deadlock-free and alive NoC. We estimate the time to develop and validate such a specification to be less than 1-man month.

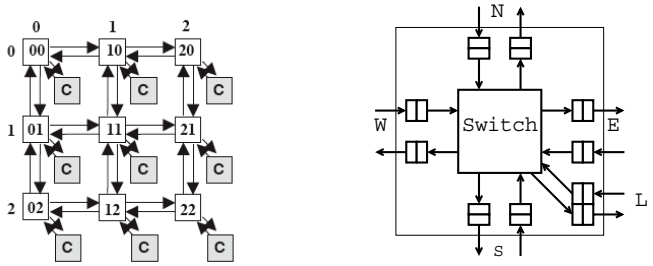
In the next Section, we briefly introduce the HERMES NoC. Section III sketches the GeNoC methodology. We present the deadlock and evacuation theorems in Section IV. Sections V and VI contains respectively the specification of the NoC and the proof of the corresponding constraints. We discuss our results in Section VII and related work in Section VIII. Finally, Section IX concludes.

II. The HERMES NoC

HERMES [9] is based on a 2D mesh architecture (Fig. 1a). Each node is made of an IP core and a switch. Each switch has five bi-directional ports: `East`, `West`, `North`, `South` connecting to the neighbor switches, and `Local` to the IP core.

The routing policy is based on a deterministic, minimal algorithm: the *XY routing algorithm*. Packets are routed first along the x-axis to the correct column, then along the y-axis to the correct processing node. HERMES uses the *wormhole* switching technique: messages are decomposed into smaller units called flits. The *header* (worm's head) flit contains information needed for routing. The *control* flit determines the number of data flits (worm's body) that follow in a pipelined fashion.

978-3-9810801-6-2/DATE10 © 2010 EDAA



(a) HERMES [9] (b) Node with 2 buffers

Figure 1. A 2D-Mesh HERMES NoC

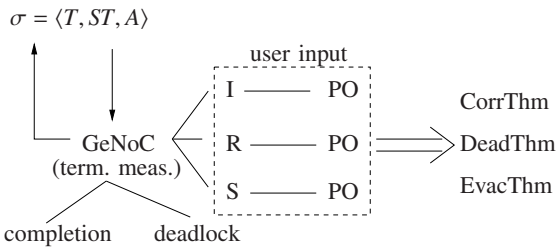


Figure 2. Specification Method Overview

We assume an HERMES network of arbitrary size. A processing node consists of *ports* and a central *switch*. There are in- and out-ports for each cardinal direction, a local in-port for injecting messages and a local out-port for removing messages from the network (see Fig. 1b). Each port has an arbitrary number of 1-flit buffers. We consider an initial list – of arbitrary size – of messages that are immediately injected in the network.

III. The GeNoC Methodology

A. Overview

Our method is illustrated in Fig. 2. At its center one finds function GeNoC which is defined by three constituents: the **I**njection method, the **R**outing function, and the **S**witching policy. These three functions are *generic*. They are not given an explicit definition but only characterized by a set of constraints or *proof obligations* (PO).

GeNoC takes as input argument an initial *configuration*, noted σ. The latter contains the list of messages to be sent on the network (T), the current value of the network state (ST), and the list of messages that have reached their destination (A). Function GeNoC recursively applies the composition of the three constituents to the initial configuration. The computation stops either when all messages have reached their destination or when the current configuration is in a deadlock. If the current configuration is not in deadlock, the switching policy must decrease the *termination measure*. This proves GeNoC always terminates.

GeNoC is characterized by three global theorems. The *Correctness Theorem* (CorrThm) states that messages reaching a destination d were actually emitted at a source node s, destined to d, and followed a valid path from s to d. The *Deadlock Theorem* (DeadThm) ensures that the routing function is deadlock-free. The *Evacuation Theorem* (EvacThm) states that all messages that have been injected in the network actually arrive at their destination and leave the network.

The proof of these theorems depends on the proof obligations associated to the constituents. It does not depend on their particular definition. Thus, these global theorems hold for all instances of the constituents that satisfy the corresponding instances of the proof obligations. This is central to the GeNoC approach and induces the following methodology.

As shown in Fig. 2, the user input consists of giving a definition to functions **I**, **R**, and **S** and to discharge the corresponding instances of the proof obligations. Once the proof obligations have been discharged, it automatically follows that the concrete instance of GeNoC satisfies the corresponding instances of the three global theorems. Thanks to the implementation of GeNoC in the ACL2 theorem proving system [10], instances of the constraints and of GeNoC are automatically generated. Moreover, the logic of ACL2 being executable, instances of GeNoC can efficiently be simulated on concrete data. In brief, from validated components a user automatically obtains a *proven correct* and *executable* specification.

B. GeNoC and its constituents

A *travel* t is a data structure which stores the progress of sending a message across a network. It is a triple $< id, c, d >$ where id is a unique identifier for the travel, c denotes the current location of the message, and d is the destination port. T denotes the list of travels that is sent across the network.

To keep the list of travels well-formed, destinations have to be reachable from their source. To this end, function $s \succ_R d$ returns true if s is reachable from d. This function is application dependent and must be instantiated. It is quite technical and not essential. We will therefore not detail it any further.

A *state* ST is a data structure which stores the current network state. The state is defined as the list of all the ports of the network. Each port is associated to the list of its buffers.

A *configuration* σ is a tuple $<T, ST, A>$, where T is a list of travels that are sent across the network, ST is a network state and A is a list of arrived travels. The travels T of configuration σ are denoted by $\sigma.T$. The set of all configurations is denoted by Σ.

Function $\mathbf{I} : \Sigma \mapsto \Sigma$ represents the *injection method*. Given a configuration, it decides which travels from T are ready for departure and injects these into the network.

Function $\mathbf{R} : P \times P \mapsto P$ represents the *routing function* of a switch. From the current position and the destination it computes the next hop. We generalize this function to apply to a configuration and to compute all hops from source to destination. We then write $\mathbf{R} : \Sigma \mapsto \Sigma$. It computes for each travel in $\sigma.T$ the route from its current location to the destination.

Function $\mathbf{S} : \Sigma \mapsto \Sigma$ represents the *switching policy*. It takes as parameter the current configuration and computes the

configuration after one *switching step*, i.e., after each message that can make progression has advanced by at most one hop. If a message arrives at its destination, the corresponding travel is removed from T and added to A.

A *deadlock-configuration* is a configuration σ in which there exists no message that can make progression. This is denoted by $\Omega(\sigma)$. An interconnection network is *deadlock-free* if and only if there exists no deadlock-configuration.

Function GeNoC is defined as follows:

$$\text{GeNoC}(\sigma) \stackrel{\text{def}}{=} \begin{cases} \sigma & \text{iff } \sigma.T = \emptyset \\ \sigma & \text{iff } \Omega(\mathbf{R}(\mathbf{I}(\sigma))) \\ \text{GeNoC}(\mathbf{S}(\mathbf{R}(\mathbf{I}(\sigma)))) & \text{iff otherwise} \end{cases}$$

IV. Deadlock and Evacuation

A. Constraints for deadlock-free routing

Dally and Seitz [11] proposed a necessary and sufficient condition for deadlock-free routing. They prove that a routing function is deadlock-free if and only if it has no cycle in its *channel* dependency graph. We have formalized a slightly different condition in GeNoC. Dally and Seitz define their function at the level of processing nodes. We define our routing function at the level of ports. We briefly present our condition and a set of constraints which are sufficient to discharge it.

Details on our formalization and its relation to Dally and Seitz' condition are available elsewhere [12]. To prove necessity, we construct a cycle from a deadlock configuration. We prove that a deadlock implies a set P of ports, which contains the next hop of each message in each port in P. The witness for P is the set of unavailable ports in the deadlock configuration. Correctness of the witness is proven by contradiction: for any message in an unavailable port, the next hop must be unavailable as well, since otherwise the message can move which contradicts the deadlock. From P we construct a graph, with P as vertices and the connected pairs of ports in P as edges. In this graph each vertex has at least one neighbor, and we prove that any such graph contains at least one cycle. Constraint (C-1), see below, ensures that the graph is a subgraph of the dependency graph. Thus the dependency graph contains a cycle.

To prove sufficiency, we construct a deadlock configuration from a cycle. Let (p_0, p_1) by a pair in the cycle. By constraint (C-2), see below, there exists a destination for messages in p_0 such that p_1 is the next hop. Each port of the cycle is filled with messages with these destinations. Since each message can have at most one next hop (determinism), all next hops of all messages are ports in the cycle. Since all ports in the cycle are filled and thus unavailable, the configuration is in deadlock.

Let $R : P \times P \mapsto P$ be a routing function. The port dependency graph is a graph with as vertices the ports of the interconnection network and as edges the pairs of ports connected by the routing function.

Theorem 1: R is deadlock-free if and only there is no cycle in its *port* dependency graph.

The proof of this necessary and sufficient condition is structured in such a way that it only depends on a fixed set of constraints over the dependency graph, the routing function, and the definition of reachable destination. Let E^R_{dep} represent the edges of the port dependency graph. These constraints are the following:

$$\forall s, d \forall p \in \mathbf{R}(s,d) \cdot s \succ_R d \implies (s,p) \in E^R_{\text{dep}} \quad \text{(C-1)}$$

$$\forall (p_o, p_1) \in E^R_{\text{dep}} \exists d \cdot p_0 \succ_R d \wedge p_1 \in \mathbf{R}(p_0, d) \quad \text{(C-2)}$$

$$\forall P' \subseteq P \cdot \neg\, \text{cycle}_{\text{dep}}(P') \quad \text{(C-3)}$$

Constraint (C-1) states that each pair of ports connected by $\mathbf{R}$ must be an edge. We consider pairs resulting from reachable destinations only. Constraint (C-2) states that for each edge (p_0, p_1) a reachable destination port must exist such that $\mathbf{R}$ routes from p_0 to p_1. Constraint (C-3) states that there is no cycle in the port dependency graph.

B. Constraints for evacuation

All messages evacuate the network if, when GeNoC terminates, the list of arrived messages equals the list of messages that were sent. This defines the Evacuation Theorem as follows:

Theorem 2: All messages eventually leave the network. Formally, we have $\text{GeNoC}(\sigma).A = \sigma.T$.

We assume all messages have been injected in the initial configuration. The injection method does not inject any more messages:

$$\mathbf{I}(\sigma) = \sigma \quad \text{(C-4)}$$

Assuming Constraints (C-1) through (C-3) are satisfied, GeNoC terminates if and only if all messages have evacuated the network. Hence, proving evacuation reduces to proving termination of function GeNoC. To prove termination, we define a *termination measure*, i.e., a value which decreases after each recursive call. Proving evacuation reduces to instantiating a function $\mu(\sigma)$, which computes a termination measure such that the following constraint holds:

$$\sigma.T \neq \emptyset \wedge \neg\, \Omega(\sigma) \implies \mu(\mathbf{S}(\mathbf{R}(\sigma))) < \mu(\sigma) \quad \text{(C-5)}$$

As long as there are messages in the network and there is no deadlock, the measure provided by μ must decrease with each switching step.

V. User Input, Part I: Executable Specification

1) Notation: A tuple <x, y, P, D> represents a port p, where x and y represent the address of the processing node of p, P is the name of p (either E, W, S, N or L) and D is the direction (either IN or OUT). Function dir(p) returns the direction of port p. Function port(p) returns the port name. Functions x(p) and y(p) return respectively the x- and y-coordinate of port p. Function trans(p, PD) returns the port specified by PD in the same processing node as p:

$$\text{trans}(p, \texttt{PD}) \stackrel{\text{def}}{=} <\text{x}(p), \text{y}(p), \texttt{P}, \texttt{D}>$$

Function next_in(p) returns the in-port connected to p, e.g., next_in(<0, 0, E, OUT>) = <1, 0, W, IN>.

2) Injection Method: As we assume that all messages are injected at time 0, the injection method is the identity function, noted $\mathbf{I}_{\mathtt{id}}$.

3) Routing Function: At each port, function $\mathbf{R}_{\mathtt{xy}}$ is defined as follows:

$$\mathbf{R}_{\mathtt{xy}}(p,d) \stackrel{\text{def}}{=} \begin{cases} \text{next\_in}(p) & \text{iff } \text{dir}(p) = \mathtt{OUT} \\ \text{trans}(p,\mathtt{WO}) & \text{iff } \text{x}(d) < x(p) \\ \text{trans}(p,\mathtt{EO}) & \text{iff } \text{x}(d) > x(p) \\ \text{trans}(p,\mathtt{NO}) & \text{iff } \text{y}(d) < y(p) \\ \text{trans}(p,\mathtt{SO}) & \text{iff } \text{y}(d) > y(p) \\ \text{trans}(p,\mathtt{LO}) & \text{iff otherwise} \end{cases}$$

4) Switching Policy: We re-use the specification of the wormhole switching policy described in Borrione *et al.* [8]. Let $\mathbf{S}_{\mathtt{wh}}$ be that function. A port accepts a flit if it has at least one available buffer. Note that a port can only accept flits of at most one packet. For each message of a configuration, function $\mathbf{S}_{\mathtt{wh}}$ moves or not the message depending on the state of the handshake protocol and available buffer spaces at the next hop.

5) Global Function: Function $\text{GeNoC}_{\mathtt{2D}}$ is the instantiation of function GeNoC. Since injection is immediate, function $\mathbf{I}_{\mathtt{id}}$ can be removed from the recursive call. Furthermore, since `xy`-routing is deterministic, the routes can be pre-computed. That is, for any configurations σ and σ', we have $\mathbf{R}_{\mathtt{xy}}(\sigma) = \mathbf{R}_{\mathtt{xy}}(\sigma')$. Therefore, function $\mathbf{R}_{\mathtt{xy}}(\sigma)$ can be removed from the recursive call as well. We extend travels to store a route as well. The route of travel t is denoted $t.r$. Given a configuration σ where all messages have been injected and routes have been pre-computed, function $\text{GeNoC}_{\mathtt{2D}}$ is defined as follows:

$$\text{GeNoC}_{\mathtt{2D}}(\sigma) \stackrel{\text{def}}{=} \begin{cases} \sigma & \text{iff } \sigma.T = \emptyset \\ \sigma & \text{iff } \Omega(\sigma) \\ \text{GeNoC}_{\mathtt{2D}}(\mathbf{S}_{\mathtt{wh}}(\sigma)) & \text{iff otherwise} \end{cases}$$

6) Dependency graph: Function next_outs(p) returns the set of out-ports connected to an in-port. It is defined as:

$$\text{next\_outs}(p) \stackrel{\text{def}}{=} \left\{ \begin{array}{lll} \text{trans}(p,\mathtt{LO}) & & \\ \text{trans}(p,\mathtt{WO}) & \text{iff} & \text{port}(p) \in \{\mathtt{E},\mathtt{L}\} \\ \text{trans}(p,\mathtt{EO}) & \text{iff} & \text{port}(p) \in \{\mathtt{W},\mathtt{L}\} \\ \text{trans}(p,\mathtt{NO}) & \text{iff} & \text{port}(p) \neq \mathtt{N} \\ \text{trans}(p,\mathtt{SO}) & \text{iff} & \text{port}(p) \neq \mathtt{S} \end{array} \right\}$$

Function $E^{\mathtt{xy}}_{\mathtt{dep}}$ is defined as follows:

$$E^{\mathtt{xy}}_{\mathtt{dep}}(p) \stackrel{\text{def}}{=} \begin{cases} \text{next\_outs}(p) & \text{iff } \text{dir}(p) = \mathtt{IN} \\ \{\text{next\_in}(p)\} & \text{iff } \text{dir}(p) = \mathtt{OUT} \end{cases}$$

It connects each out-port to the next in-port, and each in-port to the set of next out-ports (see Fig. 3).

VI. User Input, Part II: Proofs

A. Deadlock

Our objective is to prove that Theorem 1 holds for our instance of the HERMES NoC, i.e., `xy`-routing in an arbitrary large 2D-mesh. Following our methodology, it is enough to discharge constraints (C-1) to (C-3).

Proof of (C-1)$_{\mathtt{xy}}$. Proof obligation (C-1) states that for any dependency introduced by the routing function there is an edge in the dependency graph. The proof proceeds by a *systematic* case-analysis on the possible outputs of function $\mathbf{R}_{\mathtt{xy}}$. For each case, the proof is rather easy as by definition any output is a neighbour and any pair of neighbours forms an edge in $E_{\mathtt{dep}}$.

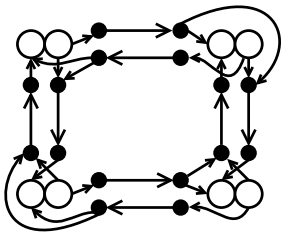


Figure 3. Port dependency graph of a 2×2-Mesh

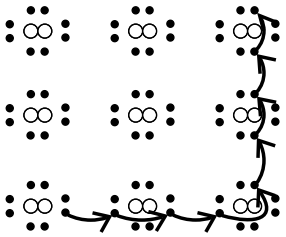


Figure 4. Northern and Western flows

Proof of (C-2)$_{\mathtt{xy}}$. This constraint states the opposite direction: any edge of the dependency graph corresponds to a possible move of the routing algorithm. We define function find_dest(p) to return the nearest destination port reachable from p:

$$\text{find\_dest}(p) \stackrel{\text{def}}{=} \begin{cases} \text{trans}(p,\mathtt{LO}) & \text{iff } \text{dir}(p) = \mathtt{IN} \\ \text{trans}(\text{next\_in}(p),\mathtt{LO}) & \text{iff } \text{dir}(p) = \mathtt{OUT} \end{cases}$$

Given a pair (p_0, p_1), we compute the nearest destination port d to p_1 and prove that $p_1 \in \mathbf{R}_{\mathtt{xy}}(p_0, d)$.

Proof of (C-3)$_{\mathtt{xy}}$. This is the main effort as we have to prove that there is no cycle in the dependency graph which is of arbitrary size. The proof proceeds by contradiction. We assume that there is a cycle. Then, we use the concept of *flows*. A flow is a sequence of ports which continually in- or decreases a coordinate. A flow is a contradiction with a cycle. There are four possible flows: one for each cardinal direction. We show that after at most one hop a flow is encountered from which there is no escape. After at most one hop it is impossible to return to the current port. Contradiction.

As example, see Fig. 4. The Northern-flow consists solely of South-In and North-Out ports. This flow continually decreases the y-coordinate. The only way to escape a Northern-flow is by entering a local out-port, which would violate the cycle assumption. As another example, the Western-flow consists solely of West-Out and East-In ports, continually increasing the x-coordinate. There are two ways to escape a Western-flow: either a local out-port, or a vertical flow. The first violates the cycle assumption, and the second has already been shown not to have an escape.

All proof obligations associated to Theorem 1 for `xy`-routing

File	Lines	Thms	Fns	CPU	Hmn
$\mathbf{R}_{xy}$	1173	97	42	16	4
$\mathbf{I}_{id}$, (C-4)	47	4	2	1	0
$\mathbf{S}_{wh}$, (C-5)	1434	151	25	17	6
$(C\text{-}1)_{xy}$	483	40	7	17	2
$(C\text{-}2)_{xy}$	435	51	0	51	2
$(C\text{-}3)_{xy}$	1018	81	10	28	4
Generic Defs	3127	234	85	2	N/A
CorrThm	2267	65	11	6	N/A
Dead/EvacThm	3277	285	125	6	N/A
Overall	13261	1008	307	144	20

Table I
OVERVIEW OF VERIFICATION EFFORT

have been discharged. Our instance of HERMES is deadlock-free.

B. Evacuation

Our objective is to prove that all messages injected in the network reach their destination, i.e., any list of pending messages can evacuate the network. As stated in Theorem 2, to prove evacuation it is enough to prove that GeNoC terminates. Following our methodology, it is sufficient to discharge constraint (C-5) for HERMES.

We need to define a termination measure and prove that it is decreased after each application of the switching policy. We define termination measure μ_{xy} to be the sum of the lengths of the routes of all messages of the configuration:

$$\mu_{xy}(\sigma) = \sum\{|m.r| \mid m \in \sigma.T\}$$

The current configuration is deadlock-free and at least one header flit can make progress. Therefore, wormhole switching (function $\mathbf{S}_{wh}$) decrements its route by 1, reducing the measure by 1 as well. Constraint $(C\text{-}5)_{wh}$ is satisfied which proves liveness of our instance of HERMES.

VII. RESULTS

Table I gives an overview of the specification and proof effort. We used the ACL2 "Sedan" [13], an Eclipse interface to the ACL2 theorem prover [10].

Column "CPU" gives the computation time to replay the definitions and the proofs on a standard laptop[1]. Numbers are counted in minutes. Column "Hmn" gives an estimate on the interaction required to perform the proof, counted in days. We assume an experienced ACL2 user familiar with the HERMES network and the GeNoC approach.

Only the upper part of table I is instantiation-specific. The generic definitions and proofs have been done *once and for all* and do not require any more human effort.

The table shows that discharging Constraints (C-1) and (C-2) takes relatively little human interaction but a long CPU time. This is because these proofs basically consist of many case distinctions. For these constraints, ACL2 provided a high degree of automation. Constraint (C-3) takes the most human interaction. We have proven this constraint for 2D-Meshes of arbitrary size. However, by instantiating the dependency graph for a fixed size network, a simple search for a cycle suffices. This search can be performed in linear time [14].

Constraint (C-5) has been proven nearly genericly, i.e., for any routing algorithm that is not both adaptive and non-minimal.

[1] 2.1 GHz Intel Core 2 Duo, MacOS X, 2 GB RAM

VIII. RELATED WORK

Several specific NoC architectures have been studied using model-checking [15], theorem proving [16] or combinations thereof [17] . Regarding formal proofs of deadlock prevention, Gebremichael *et al.* [16] formally prove a sufficient condition for the Æthereal protocol of Philips in a packet-switched network. The main property that has been verified is the absence of deadlock for an arbitrary number of masters and slaves. These works target very specific design described at a low level of abstraction. Our method supports the specification of a large class of systems described at a higher-level of abstraction. More recently, Taktak *et al.* [18] defined an algorithm checking a condition sufficient for deadlock-freedom for adaptive routing in NoCs. This work focuses on deadlock detection and first extracts the strongly connected components of the dependency graph. Then, it looks for cycles between these components. This approach could be used to discharge constraint (C-3) for fixed instances.

IX. CONCLUSION

We presented a specification and validation methodology for high-level and *parametric* descriptions of NoC designs. Our method includes the proof that the network is deadlock-free and can evacuate all injected messages. The proof is fully parametric: the number of messages, their size, the number of buffers at each node are left uninterpreted. Thanks to the use of the ACL2 theorem prover, our models are also executable. The same models are used for both formal validation and simulation.

Our method currently applies to deterministic routing algorithms. This restriction is due to our deadlock condition. We plan to formalize such a condition for adaptive routing [19], [20], [21]. The main tasks will be to define a different dependency graph and formally check the condition. The rest of the method would not be affected. Another restriction of our current model is that all messages must be injected immediately. Regarding evacuation, Theorem 2 could be rephrased such that it proves that all *injected* messages reach a destination, i.e., all injected messages eventually leave the network. We are working on the proof that all messages are *eventually* injected. This proof entails a generic bound on the injection time of each message. This injection time is not only dependent on the injection method, but also on the state of the network, e.g., the latency and the number of injection buffers. Deadlock-freedom is necessary, since otherwise there is no guarantee that an unavailable injection buffer eventually becomes available.

The GeNoC model used in this paper is at the specification level. Recently, van den Broek and Schmaltz [22] defined two variations (implementation and specification) of the GeNoC

model and formally proved them equivalent. We plan to integrate our deadlock and evacuation theorems to these new models and their relation contributing to a general cross-layer verification method for NoCs.

Acknowledgments

We would like to thank the anonymous reviewers for their apposite, constructive, and detailed comments. This research is supported by NWO/EW project Formal Validation of Deadlock Avoidance Mechanisms (FVDAM) under grant no. 612.064.811.

References

[1] W. Dally, "The end of denial architecture," Keynote at Design Automation Conference (DAC'09), 2009.

[2] J. van Meerbergen, "Networks on chip: A communication-centric approach to platform-based design," in *PROGRESS White Papers 2006*. STW, The Netherlands.

[3] W. J. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," in *Proceedings of the Design Automation Conference*, Las Vegas, NV, 2001, pp. 684–689.

[4] L. Benini and G. D. Micheli, "Networks on Chips: A New SoC Paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, 2002.

[5] J. Schmaltz and D. Borrione, "A functional formalization of on chip communications," *Formal Aspects of Computing*, vol. 20, pp. 241–258, 2008.

[6] D. Borrione, A. Helmy, L. Pierre, and J. Schmaltz, "A formal approach to the verification of Networks on Chip," *EURASIP Journal on Embedded Systems*, vol. 2009, 2009, article ID 548324.

[7] J. Schmaltz and D. Borrione, "Towards a Formal Theory of On Chip Communications in the ACL2 Logic," in *Proceedings of the Sixth International Workshop on the ACL2 Theorem Prover and its Applications, part of FloC'06*. Seattle, Washington, USA: ACM, August 14-15 2006.

[8] D. Borrione, A. Helmy, L. Pierre, and J. Schmaltz, "Executable formal specification and validation of NoC communication infrastructures," in *Proceedings of the 21st annual symposium on Integrated circuits and system design (SBCCI'08)*. Gramado, Brazil: ACM, September 1–4 2008, pp. 176–181.

[9] F. Moraes, N. Calazans, A. Mello, L. Möller, and L. Ost, "Hermes: an infrastructure for low area overhead packet-switching networks on chip," *Integration, the VLSI Journal*, vol. 38, no. 1, pp. 69 – 93, 2004.

[10] M. Kaufmann, P. Manolios, and J. S. Moore, "ACL2 Computer-Aided Reasoning: An Approach," 2000.

[11] W. Dally and C. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Transactions on Computers*, no. 36, 1987.

[12] F. Verbeek and J. Schmaltz, "Proof pearl: A formal proof of Dally and Seitz' necessary and sufficient condition for deadlock-free routing in interconnection networks," submitted to publication. Available on-line at www.cs.ru.nl/~julien/Julien_at_Nijmegen/JAR09.html.

[13] P. C. Dillinger, P. Manolios, D. Vroon, and J. S. Moore, "ACL2s: The ACL2 sedan," *Electronic Notes in Theoretical Computer Science*, vol. 174, no. 2, pp. 3 – 18, 2007, proceedings of the 7th Workshop on User Interfaces for Theorem Provers (UITP 2006).

[14] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*. MIT Press and McGraw Hill, 1990.

[15] A. Roychoudhury, T. Mitra, and S. Karri, "Using Formal Techniques to Debug the AMBA System-on-Chip Bus Protocol," in *Design Automation and Test Europe (DATE'03)*, 2003, pp. 828–833.

[16] B. Gebremichaeli, F. Vaandrager, M. Zhang, K. Goossens, E. Rijpkema, and A. Rădulescu, "Deadlock prevention in the Æthereal protocol," *Correct Hardware Design and Verification Methods*, vol. 3725/2005, pp. 345–348, 2005.

[17] H. Amjad, "Model Checking the AMBA Protocol in HOL," University of Cambridge, Computer Laboratory, Tech. Rep., September 2004.

[18] S. Taktak, J.-L. Desbarbieux, and E. Encrenaz, "A tool for automatic detection of deadlock in wormhole networks on chip," *ACM Transactions on Design Automation of Electronic Systems*, vol. 13, no. 1, January 2008.

[19] J. Duato, "A necessary and sufficient condition for deadlock-free adaptive routing in wormhole networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, no. 10, pp. 1055–1067, 10 1995.

[20] L. Schwiebert and D. N. Jayasimha, "A universal proof technique for deadlock-free routing in interconnection networks," in *In 7th Annual ACM Symposium on Parallel Algorithms and Architectures*, 1995, pp. 175–184.

[21] E. Fleury and P. Fraigniaud, "A general theory for deadlock avoidance in wormhole-routed networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 7, pp. 626–638, 1998.

[22] T. van den Broek and J. Schmaltz, "Towards formally verified networks-on-chips," in *Proceedings of Formal Methods in Computer Aided Design (FM CAD'09)*. Austing, Texas, USA: IEEE Computer Society, November 2009.