A Tool for Automatic Detection of Deadlock in Wormhole Networks on Chip

SAMI TAKTAK, JEAN-LOU DESBARBIEUX
LIP6 Lab – University Paris VI & CNRS
and
EMMANUELLE ENCRENAZ
LSV – ENS Cachan & CNRS

We present an extension of Duato's necessary and sufficient condition a routing function must satisfy in order to be deadlock-free, to support environment constraints inducing *extra-dependencies* between messages. We also present an original algorithm to automatically check the deadlock-freeness of a network with a given routing function. A prototype tool has been developed and automatic deadlock checking of large scale networks with various routing functions have been successfully achieved. We provide comparative results with standard approach, highlighting the benefits of our method.

Categories and Subject Descriptors: B.4.3 [**Input/Output and Data Communications**]: Interconnections (Subsystems)—*topology*; D.2.4 [**Software**]: Software/Program Verification—*Formal methods*

General Terms: Algorithms, Verification

Additional Key Words and Phrases: Deadlock, Interconnection networks, Networks on Chip, Wormhole routing

1. INTRODUCTION

Networks on chip (NoC) are a critical part of System on chip (SoC). Indeed, the growing size of SoC including many components, requires the use of distributed network [Dally and Towles 2001]. The interconnect introduces latency in communication between components. So wormhole routing is often used since it significantly reduces the latency of the network and avoids using large storage buffers in the routers. We can find in [Ni and McKinley 1993] a review of wormhole routing techniques.

Deadlock is an important problem for a network on chip design. In practice,

Author's address: S. Taktak (sami.taktak@lip6.fr) and J.L. Desbarbieux (jean-lou.desbarbieux@lip6.fr), Laboratoire d'Informatique de Paris 6, Universite Pierre et Marie Curie, 4, Place Jussieu, 75252 Paris Cedex 05, France.
E. Encrenaz (emmanuelle.encrenaz@lsv.ens-cachan.fr), Laboratoire Specification et Verification, CNRS UMR 8643, Ecole Normale Superieure de Cachan, 61, avenue du President Wilson, 94235 CACHAN Cedex, France

network designers are used to duplicate hardware to avoid resource sharing that may induce deadlocks. The cost of this simple solution can be prohibitive but alternative solutions, requiring less hardware, induce questionings about deadlocks that have to be solved.

A lot of work has been done to determine if a network is deadlock free: during the last decade, different approaches have been investigated to analyze deadlocks. Those based on *dependency graphs analysis* are the most commonly used.

Dally and Seitz [1987] gives the necessary and sufficient condition for a *deterministic* routing algorithm to be deadlock-free. This condition states that a deterministic routing function is deadlock-free iff there is no cycle in its *channel dependency graph*. He also shows how we can construct a deadlock free-routing function for an arbitrary network by introducing virtual channels.

Duato [1995] provides a necessary and sufficient condition for an *adaptive* routing network to be deadlock-free. The adaptive function depends on the current node and on the destination. This condition allows the existence of cyclic dependencies between channels, represented in the *channel dependency graph*, but a routing subfunction must exist and it must have no cycle in its channel dependency graph (a routing subfunction is a restriction of the routing function). This permits a design with minimum restrictions and as few virtual channels per physical channel as possible. This is an important point, since virtual channels are expensive in hardware and increase node delay [Chien 1998].

Fleury and Fraigniaud [1998] proposed a general theory for the study of routing in wormhole-routed networks. This theory applies to a wide class of routing functions and includes most of the definitions of routing functions: compact routing, vertex-dependent, input-dependent, source-dependent, history-dependent, path-dependent, multi-dependent, library-dependent.

Schwiebert and Jayasimha [1996] introduce a new necessary and sufficient condition for deadlock-free wormhole routing considering adaptive functions depending on the current input channel and on the destination. They introduce the *channel waiting graph*. In [Jayasimha et al. 2003], Schwiebert and Jayasimha have extended their theory to support a larger class of routing functions.

All the above theories assume that a message arriving at its destination is eventually consumed. As example, in the SPIN/VCI [Charlery et al. 2004] network this condition is not satisfied: some deadlocks may occur due to the dependencies between different kinds of messages. VCI protocol defines two types of messages: request and responses. Each request message must be acknowledged by a response message of (at least) the same size. The destination node of a request message is also the injection node of the response message. As internal buffer of each node is bounded, the destination node may evacuate a new cell of the request on its delivery channel only if it can consume at the same time a cell of the response message on its injection channel.

Thus, this introduces a dependency between delivery channel and injection channel that is not considered in the works of Dally and Seitz [1987], Duato [1995], Fleury and Fraigniaud [1998], Schwiebert and Jayasimha [1996], Jayasimha et al. [2003].

Using a deadlock-free interconnect and a deadlock-free network protocol do not imply that the resulting network will be deadlock-free. A standard solution is to use separate channels for request and response messages which generally implies a duplication of resources.

In this paper we propose an adaptation of Duato's theory to take into account the dependencies introduced by the environment of the network without the need of using dedicated channel. We provide a large condition allowing channels to be shared by request and response messages.

In the previous example, we distinguish two kinds of message: *request* and *response* messages. We will say that there are two *types* of message. As the progression of a message of type request depends on the progression of a message of type response, we will say that a message of type response has a higher priority than a message of type request. Then, to prove that such a network transmitting these types of messages, is deadlock-free, we have to prove that messages of type response can always be delivered, regardless which messages of type request are in the network. Then, we have to deal with messages of type request. If we can show that messages of type request can also be delivered for any valid configuration, we have shown that the network is deadlock-free. That is, we introduce the notion of *type* of messages to represent the dependencies due to the environment and those dependencies induce an order on messages types. If messages can be evacuated following the message type order, the network is deadlock-free.

All techniques for proving deadlock-freeness are based on search and elimination of cycles in the extended dependency graph and require exponential time in worst case [Schwiebert and Jayasimha 1996], [Duato 1991], [Duato 1995], [Lin et al. 1995], [Ianni 1997]. Here we will present a new approach to check if a network is deadlock-free based on *Strongly Connected Component* (SCC) analysis of the extended dependency graph. We will also propose a methodology to suppress those strongly connected components, preserving the connectedness of the routing function. *This technique avoids to check for connectedness.*

The condition to reduce a strongly connected component is sufficient but not necessary. Hence our methodology is conservative but may not reduce strongly connected components that do not effectively involve deadlocks. In practice, all the deadlocks detected by our tool were real deadlocks. We experiment the standard cycle based approach, (as described in [Schwiebert and Jayasimha 1996], [Fleury and Fraigniaud 1998]) with our method based on SCC, and we provide comparative results on a set of networks enlightening the effectiveness of SCC-based approach.

The remainder of the paper is organized as follows. Section 2 presents an extension of Duato's necessary and sufficient condition to represent the dependencies between injection and delivery channels due to the environment. Section 3 presents a sufficient condition to eliminate cycles that do not lead to a deadlock. Section 4 presents an original algorithm based on this condition to determine automatically if a network is deadlock-free. Section 5 presents the application of our tool on a set of real networks on chip [Charlery et al. 2004], [Panades et al. 2006] with different routing strategies to detect potential deadlock, and comments the results. Then we conclude and sketch future directions of work.

2. DEADLOCK ANALYSIS OF NETWORKS ON CHIP

2.1 Networks on Chip specificities

Networks on Chip (NoC) is aimed to replace buses on SoC. In fact, despite many efforts to improve buses performances, these can not address today's issues in term of scalability and bandwidth. Since NoC is embedded on SoC [Dally and Towles 2001], they must remain simple due to surface and power consumptions considerations [Kim et al. 2005]. Also, simple routing function permit small latency on the NoC. Mesh is a commonly used topology associated with a X-Y routing function [Bjerregaard and Sparso 2005], [Goossens et al. 2005]. The latency of a NoC is of the order of hundreds of processor cycles. For these reasons, NoC differ from traditional networks. The topologies commonly used in a NoC are *regular* and *fixed* [Bjerregaard and Sparso 2005]: there is no hot-plug capability. Typical throughput are of some decades of gigabit per seconds. ×pipes [Stergiou et al. 2005], Mango [Bjerregaard and Sparso 2005], Æthereal[Goossens et al. 2005] or QNoS [Bolotin et al. 2004] are examples of on-chip network. However, with the growing size of SoC which contains hundreds of nodes, failures of routers due to fabrication process are more common. Using SoC with some defective nodes maintain a high level of production, while reducing production cost. The price to paid is to support more complex routing functions [Bolotin et al. 2007] that deal with defective nodes and irregular topologies. Designers are faced to determine whether a complex routing function is deadlock-free or not.

2.2 Global hypothesis

The interconnect is a collection of routers connected by channels. Each router can send messages, transmit messages from one of its input channel to one of its output channel according to the routing function, or it consumes a message if this latest has reached its destination. We make no distinction between "packet" (in the VCI terminology) and "message" (current terminology in deadlock-free network studies). In this document, we use the term of "message". The unbreakable transfer unit is called *flit*.

The following hypothesis are mainly those used in [Duato 1995]. Some have been modified or added to take into account the type of messages (hypothesis 1, 2, 6).

(1) The messages set is split into disjoint sets of typed messages ordered by decreasing priority. Let t and t' two types of messages, if $t < t'$, we says that t has a higher priority than t'.

(2) When a message arrives at its destination, it can be consumed *under conditions*. Only message of highest priority type has to be consumed without any condition.

(3) A node can generate a message of any length destined for any other node on the network.

(4) Wormhole routing is used. So when a channel accepts a message, it must accept the remaining of the message before it accepts any other message. A message may occupy several channels at the same time.

(5) A channel cannot contain flits belonging to different messages at the same time. Thus, a blocked message has always its head on the top of a channel.

(6) The path followed by a message depends of its destination, its *type* and of the state of output channels of the current node. At each node, an *adaptive routing function* gives a set of output channels for a given message depending on its *type*, its destination and the current node. A *selecting function* selects a free output channel within those given by the routing function. If all output channels are busy, the message waits until an output channel becomes free.

(7) All messages arriving at a node are processed in parallel.

(8) When several messages are waiting for a free output channel, they are proceed in an order that prevents starvation.

2.3 Definitions

This section defines precisely the network topology and routing function. The definitions are mostly taken from [Duato 1995] and are here since we need them to present our work. Some have been adapted, and others were added, to take into account the message type.

Definition 2.1. An *interconnect network* I is a strongly connected directed multigraph, $I = G(N, C)$. The nodes of the multigraph N represent the routers of the network. The edges of the multigraph C represent the channels of the network. Multiple edges between nodes are permitted, but no self loop is allowed. The node source (resp. destination) of a channel c is named $s(c)$ (resp. $d(c)$).

Definition 2.2. $F = \{\text{free}, \text{busy}\}$ is the *set of valid states* of a channel.

Definition 2.3. A message is made of a sequence of *flits*. A *flit* is the unbreakable transfer unit.

Definition 2.4. T is the *set of types of message* that can transit on the network. They are ordered such as:
$\forall\, t_1, t_2 \in T,\ t_1 < t_2$ if the progression of a message of t_2 depends on the progression of a message of type t_1. $<$ is a partial order.

In other words, a message of type t_2 can progress only if a message of type t_1 can progress.

Definition 2.5. A *message* is represented as a pair in $N \times T$ defining its destination and its type.

Definition 2.6. $label(c)$ is a set of pairs in $N \times T$, each of which represents a message that can be sent through the channel c.

Definition 2.7. An *adaptive routing function* $R : N \times N \times T \rightarrow \mathcal{P}(C)$, where $\mathcal{P}(C)$ is the power set of C, supplies a set of channels to send a message of type t from the current node n_c to the destination node n_d, $R(n_c, n_d, t) = \{c_1, c_2, \ldots, c_p\}$. By definition, $R(n, n, t) = \emptyset,\ \forall\, n \in N,\ \forall\, t \in T$.

Definition 2.8. A *selection function* $S : \mathcal{P}(C \times F) \rightarrow C$ selects a free output channel from those supplied by the routing function. S takes into account the state of the channel supplied by the routing function. The selection function should avoid starvation. If all output channels are busy, the message waits until an output channel becomes free.

Definition 2.9. A routing function R for an interconnection network I is *connected* iff:

$$\forall\, t \in T,\ \forall\, x, y \in N,\ x \neq y,\ \exists\, c_1, c_2, \ldots, c_k \in C$$

such as:

$$\begin{cases} c_1 \in R(x, y, t) \\ c_{m+1} \in R(d(c_m), y, t),\ m = 1, \ldots, k-1 \\ d(c_k) = y \end{cases}$$

So, a function is connected if one can find a path $P(x, y)$ from x to y using channels provided by R, for any x and y and any type t.

Definition 2.10. A *routing subfunction* R_1 of a given routing function R is a routing function which supplies a subset of the channels supplied by R:

$$R_1(x, y, t) \subseteq R(x, y, t),\ \forall\, x, y \in N,\ \forall\, t \in T.$$

We define also the complementary function R_1^R

$$R_1^R = R(x, y, t) \setminus R_1(x, y, t),\ \forall\, x, y \in N,\ \forall\, t \in T$$

Definition 2.11. A *configuration* is a set of flits assigned to each channel of the interconnect. The numbers of flits in a channel c_i is noted $size(c_i)$. A message is present on a channel if a flit of this message is present on this channel. The destination of a message m_i is denoted $dest(m_i)$.

Definition 2.12. A *valid configuration* is a configuration that can be reached from an empty network which is filled with respect to the routing function.

Definition 2.13. A *deadlock configuration* is a nonempty configuration where no message can progress.

Definition 2.14. A routing function R for an interconnect I is deadlock-free iff there is no valid deadlock configuration for this routing function.

Definition 2.15. For a given interconnect I, a set of messages' types T, a given routing function R, a routing subfunction R_1 of R and two channels $c_i, c_j \in C$:

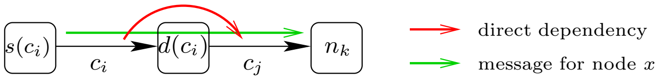


Figure 1

— There is a *direct dependency* from c_i to c_j iff

$$\exists\, x \in N,\ \exists\, t \in T \text{ such as}$$

$$c_i \in R(s(c_i), x, t) \text{ and } c_j \in R(d(c_i), x, t)$$

There is a direct dependency from c_i to c_j iff there is a message in c_i that can be forwarded to c_j. c_i and c_j are supplied by R for that message.

— There is an *indirect dependency* from c_i to c_j iff

$$\exists\, x \in N,\ \exists\, t \in T,\ \exists c_1, c_2, \ldots, c_k \in C \text{ such as}$$

$$c_i \in\ R_1(s(c_i), x, t),\ \ c_j \in R_1(d(c_k), x, t)$$

$$c_1 \in\ R_1^R(d(c_i), x, t) \text{ and } c_m \in R_1^R(d(c_{m-1}), x, t),$$

$$m = 2, \ldots, k$$

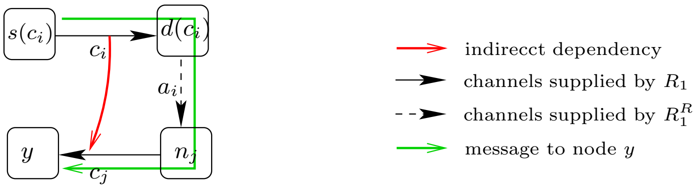


Figure 2

There is an indirect dependency from c_i to c_j iff there is a message in c_i that can be forwarded to c_j via channels not supplied by R_1 for that message. c_i and c_j are supplied by R_1 for that message.

— There is a *direct cross dependency* from c_i to c_j iff

$$\exists\, x, y \in N,\ \exists\, t, t' \in T \text{ such as}$$

$$\begin{cases} c_i \in R_1(s(c_i), x, t), \\ c_i \in R_1^R(s(c_i), y, t'), \\ c_j \in R_1(d(c_i), y, t') \end{cases}$$

There is a direct cross dependency from c_i to c_j iff there is a message in c_i that can be forwarded to c_j. c_i is not supplied by R_1 for that message but c_j is.

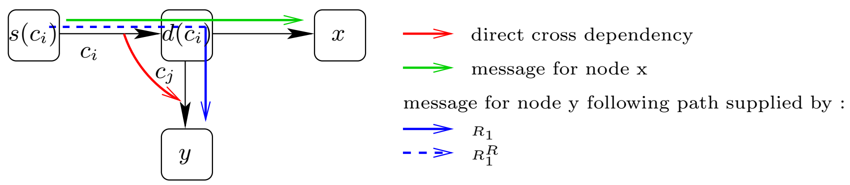


Figure 3

— There is an *indirect cross dependency* from c_i to c_j iff

$$\exists\, x, y \in N,\ \exists\, t, t' \in T,\ \exists c_1, c_2, \ldots, c_k \in C \text{ such as}$$

$$\begin{cases} c_i \in R_1(s(c_i), x, t), \\ c_i \in R_1^R(s(c_i), y, t'), \\ c_j \in R_1(d(c_k), y, t'), \\ c_1 \in R_1^R(d(c_i), y, t'), \\ c_m \in R_1^R(d(c_{m-1}), y, t'), m = 2, \ldots, k \end{cases}$$

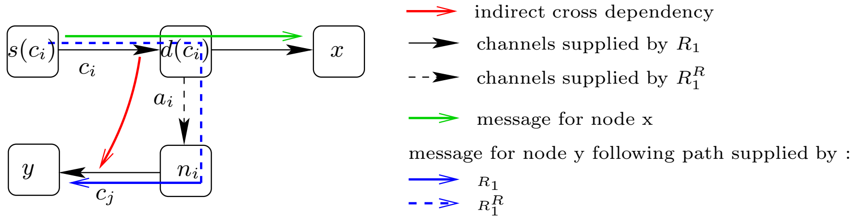


Figure 4

There is an indirect cross dependency from c_i to c_j iff there is a message in c_i that can be forwarded to c_j via channels not supplied by R_1 for that message. c_i is not supplied by R_1 for that message but c_j is. c_i is supplied by R_1 for some other messages.

Definition 2.16. An *Extended Dependency Graph* (EDG) associated with a routing function R_1 is a directed graph. Its vertexes are the channels of the interconnect. Its edges are the direct, indirect, direct cross and indirect cross dependencies induced by R_1 relatively to R between channels.

Theorem 2.17 Duato's theorem [Duato 1995]. *A coherent, connected and adaptive routing function R for an interconnection network I is deadlock-free iff there exists a routing subfunction R_1 that is connected and has no cycle in its extended channel dependency graph.*

Definition 2.18. R_t is a routing subfunction of R defined by:
$\forall\, x, y \in N,\ \forall\, t, t' \in T,\ t < t'$

$$R_t(x, y) \subseteq R(x, y, t)$$

such that

$$\forall c \in R_t(x, y), c \notin R(x, y, t')$$

That is, the channel supplied by R_t are only used by messages of type t or by messages of higher priority type.

THEOREM 2.19. *Let R be a connected and adaptive routing function and T an ordered set of message types. If $\forall\, t \in T,\ R_t$ is connected and deadlock free, then R is deadlock free.*

R_t supplies an escape path for each message of type t. Let t_I be the highest type of message. If we have R_{t_I} deadlock free, then all messages of type t_I can be evacuated of the interconnect by using channel supplied by R_{t_I}. Then any message of type $t_I < t_n$ and $\nexists\, t_k | t_I < t_k < t_n$, can be evacuated using channels supplied by R_{t_n}. Channels supplied by R_{t_n} may only contain messages of type t_I or messages of type t_n. Since all messages of type t_I can be evacuated by channels supplied by R_{t_I}, we can remove from the interconnect all messages of type t_I. Then, if R_{t_n} is deadlock-free, all messages of type t_n can be evacuated.

PROOF. *a) Let $t \in T \mid \forall\, t' \in T,\ t < t'$. We assume that R_t is connected and deadlock-free. Supposing there is a deadlock configuration for R involving a message m_t of type t. There are two cases:*

—*the head of m_t is on a channel supplied by R_t, then R_t is not deadlock-free, which is breaking our hypothesis.*
—*the head of m_t is not on a channel supplied by R_t. Let x be the node where the head of m_t is and let y be its destination. Since R_t is connected, $\exists\, c \in R(x, y, t)$ such as $c \in R_t(x, y)$. Thus this message can take a path supplied by R_t, R_t is deadlock-free, so this message can progress in the interconnect.*

So, $R(x, y, t)\ \forall\, x, y \in N$ is deadlock-free. Now we need to prove that $R(x, y, t')$ is deadlock-free for any t'.

b) Suppose $R(x, y, t)$ deadlock-free for any $t < t', t' \in R$. We have to show that $R(x, y, t')$ is deadlock free. We suppose there exists a deadlock configuration for R involving a message $m_{t'}$ of type t'. There are two cases:

—*if $m_{t'}$ is on a channel supplied by $R_{t'}$, then $m_{t'}$ is blocked by a message of type t' or by a message m_t of type t with $t < t'$. By hypothesis, R is deadlock-free for all messages of type t, $t < t'$. Hence m_t is not blocked and can progress on the interconnect and m'_t will progress. If $m_{t'}$ is blocked by a message of type t', then R'_t is not deadlock-free which is in opposition with the hypothesis on $R_{t'}$.*
—*if $p_{t'}$ is not on a channel supplied by $R_{t'}$. Let x be the node where the head of m'_t is and let y be its destination. In this case, $\exists c \in R(x, y, t')$ such as $c \in R_{t'}(x, y)$ since $R_{t'}$ is connected. But $R_{t'}$ is deadlock-free, so this message is not blocked.*

We have shown that no message can be involved in a deadlock configuration, thus R is deadlock free.

□

3. A SUFFICIENT CONDITION GUARANTYING DEADLOCK-FREENESS

Given a routing function R, an interconnect I and an ordered set of messages types, to prove R to be deadlock-free, we seek a routing subfunction for each R_t that has no cycle in its EDG.

Our approach consists in finding all Strongly Connected Component (SCC) of the extended channel dependency graph, and to work on it instead of working on each cycle of the EDG as suggested in [Schwiebert and Jayasimha 1996]. Finding all SCC can be done in polynomial time. Then we will try to suppress each SCC by restricting the routing function. This reduction preserves the connectedness of the network so we do not have to check if the function remains connected after removing a channel. If we can remove all SCC, then we have found a routing subfunction that is connected and has an acyclic channel dependency graph. Thus the network is deadlock-free.

The condition below will ensure that a cycle will not lead to a deadlock. It is a sufficient condition.

The idea is to find in a cycle, a channel c that can be emptied regardless which messages are present in the cycle. To find such a c, we look at each channel of the cycle. Let c_0 be a channel of the cycle and c_k be the latest channel of the cycle that can receive a message from c_0. Then we look if any messages that come from c_0, in channels c_0 to c_k can be forwarded to channels not belonging to the cycle.

If we can find such channels, these messages cannot be involved in a deadlock and this cycle will not produce deadlock. So we can construct a routing subfunction R_1 of R_t such that any messages transiting through c_0 cannot be transmit to any c_i $(i = 1, \ldots, k)$. R_1 will not have this cycle in it's EDG.

An example is given at the end of this section.

Condition 3.1 *A sufficient condition to suppress cycles.* If, for a routing function R and an interconnect I, there is a cycle such as:
$\exists\, c_0, \ldots, c_k \in \mathit{cycle}$, such as

$$\begin{cases} \exists\ (n,t) \in \bigcap_{j=0}^{m}\ \mathit{label}(c_j) \mid c_m \in R(d(c_{m-1}), n, t), \\ \bigcap_{j=0}^{k}\ \mathit{label}(c_j) = \emptyset \text{ or } c_k = c_0, \text{ for all } m = 1, \ldots, k-1 \end{cases}$$

and $\exists\, Y_0, \ldots, Y_k$ sets of channels not in cycle such as

$$\begin{cases} \forall\ (n,t) \in \bigcap_{j=0}^{m}\ \mathit{label}(c_j), \\ \exists\ y_m \in Y_m \mid y_m \in R(d(c_m), n, t), \text{ for all } m = 0, \ldots, k-1 \end{cases}$$

then this cycle will not lead to a deadlock configuration.

Proof. Let us consider a cycle C made of channels $c_i, i = 0, \ldots, k$ which satisfy condition 3.1.

By definition, a deadlock configuration is a nonempty configuration where no message can progress.

Let us suppose there is a deadlock configuration D_C for this cycle. Since there is a deadlock in C, c_0 is not empty and contains a message m of type t. We assume any channel not belonging to C is not involved in the deadlock and can be emptied.

Let c_l, the channel containing the head of m.

Since D_C is a deadlock configuration, m can not progress in the network.
Condition 3.1 ensures that

$$\forall\ (n,t) \in \bigcap_{j=0}^{m} label(c_j), \exists\ y_m \in Y_m \mid y_m \in R(d(c_m), n, t),\ \text{for all } m = 0, \ldots, k-1.$$

So, $\exists\ y_l$ such as $y_l = R(d(c_l), dest(m), t)$ and $y_l \notin C$. Since, y_l do not belong to C, y_l is not involved in the deadlock and can be emptied.
m can be forwarded by $d(c_l)$ to y_m.
m can progress in the network.
D_C is not a deadlock configuration. $\square$

We do not apply this condition to each cycle in the EDG. Instead, we apply it to strongly connected components of the EDG. Since a SCC is a collection of cycles sharing some edges, the same reasoning apply. The y_i channels of condition 1 must be out of the SCC.

Thus we can restrict R to obtain R_1 by not allowing messages that may be sent through c_0 to transit through $c_i (i = 1, \ldots, k)$. This reduction does not create new cycle. New dependencies in the EDG of R_1 will be either indirect dependencies from channels $a_i \in SCC$ such as $c_0 \in R(d(a_i), dest(m), t)$ and channels b_i not in SCC, or (indirect) cross dependencies, between $c_i (i = 1, \ldots, k)$ and channels b_i. As there is no path from b_i to a_i, nor from b_i to c_i, since a_i and c_i do not belong to the same SCC than b_i. Then no cycle can be created.

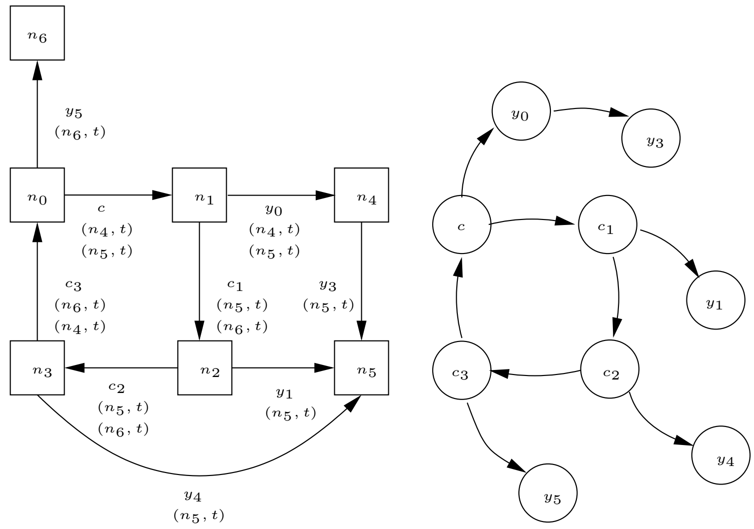
In Figure 5(a), we can see an interconnect with seven nodes. Each channel is labeled with a name and the set of pairs (destination, type) of the messages that can be sent over this channel. The extended dependency graph of this network is presented on Figure 5(b). It contains a cycle made by the channels c, c_1, c_2 and c_3. But there is no deadlock configurations for this network: considering the channel c, any message in the channel c can be delivered

—If there is the head of a message (n_4, t) in c, it will be routed to y_0 and reach its destination.

—If there is a message (n_5, t) in c:
  —if its head is in c. If c_1 is busy, (n_5, t) will be routed through y_0.
  —If its head is in c_1 and if c_2 is busy, it will be routed through y_1.
  —Finally, if its head is in c_2, it will be routed through y_4

Thus, the channel c can always be emptied and no deadlock can occur. Then we can restrict the routing function by forbidding the transit of messages (n_5, t) from c to c_1, and we have a connected routing subfunction with an acyclic EDG. Thus, this network is deadlock-free.

If we consider the same network without the channel y_1, we can easily find a deadlock configuration (see Figure 6). Such a cycle does not satisfy condition 3.1, hence it cannot be suppressed. The SCC is not reducible, the network is not guaranteed to be deadlock-free.



(a) The interconnect and a routing functionR_t

(b) The extended dependency graph

Fig. 5. An example of cycle where a deadlock cannot occur

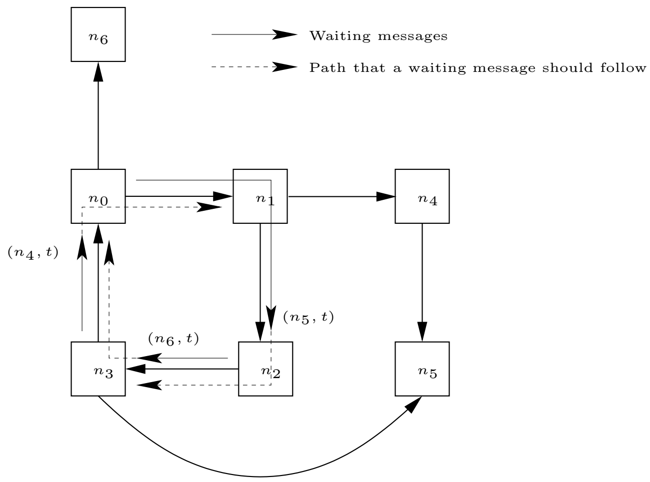


Fig. 6. A deadlock configuration

4. AUTOMATIC PROOF OF DEADLOCK FREE NETWORK

4.1 Deadlocks detection

In this section, we describe a new algorithm based on theorem 2.19 and condition 3.1 to prove automatically that a network is deadlock-free.

The topology of the interconnect I, the Routing function R and the ordered set of messages types are supplied. The internal data structure represents the interconnection graph I as a list of nodes and a list of channels. Each channel is related to the set of messages that may be sent through it.

To check if the interconnect is deadlock-free, we proceed as follows:

—split R into subfunctions R_t according to T
—for each t in T:
  (1) check for connectedness of R_t
  (2) construct the EDG of R_t
  (3) find all SCC of R_t
  (4) break each SCC using condition 3.1
—if all SCC have been broken, the network is deadlock-free. Else it may not be deadlock-free: print the irreducible SCC.

To construct the extended dependency graph, we find all dependencies of each arcs.

Step 2: Construction of $EDG(R_t)$

INPUTS: the subfunction R_t
for each arc a **do**
  add a to the dependency graph
endfor
for each node n of the dependency graph **do**
  for each outgoing arc a of n **do**
    add each direct dependency of a to the $EDG(R_t)$
    add each indirect dependency of a to the $EDG(R_t)$
  endfor
endfor

Step 3: Finding SCC in $EDG(R_t)$

We use the well known algorithm of Tarjan [Knuth 1993] that returns the set of SCC of $EDG(R_t)$.

Step 4: Break all SCC of $EDG(R_t)$

Now we will try to find a routing subfunction by reducing the given routing function. To be able to reduce the routing function, one has to check if each message along each cycle can be evacuated through an escape path, assuming any configuration of the current SCC.

This algorithm is composed of three functions. The main function tries to reduce each SCC. To reduce a SCC, `suppress_cycle` is called. `suppress_cycle` call `exists_escape_path` to check if a message in a channel of the SCC can be forwarded to a channel out of the SCC.

INPUTS: set of strongly connected component SCC of the EDG of R_t
for all $C \in$ SCC **do**
  if it's not reduced to one node **then**
    call:Suppress_Cycle(C)
  endif
endfor

suppress_cycle(C)***:***
INPUTS: a SCC C
for all nodes v of the C **do**
  if call:Exists_Escape_Path (v, $labels(v)$) **then**
    remove v from the C
  endif
endfor

Exists_Escape_Path(v, l)***:***
INPUTS:node v of the SCC C
  set of labels l of v

let $nlechap \leftarrow \{(d,t)|(d,t) \in l \text{ and } \forall u \notin C, d(v) \neq s(u)\}$
// lnechap is the subset of l whose message cannot be evacuated
// through an escape node leaving the current SCC C.
$l \leftarrow l \setminus \{(d,t)|d = v\}$
// remove labels of messages having reached their destination

if $nlechap \neq \emptyset$ **then** *// a deadlock is possible*
  return NO
else
  for all $u \in C$ such that $d(v) = s(u)$ **do**
    if call:Exists_Escape_Path(u,$l \cap labels(u)$) == NO
      return NO
    endif
  endfor
  return YES
endif

4.2 Example

In this section, we present an example showing how the algorithm works. For simplicity, this example contains only one type of messages. If we had more message's types, we would apply the same methodology to each routing subfunction R_t in the order of the priority of the types.

The interconnect is shown on Figure 7 where all edges are labeled. This interconnect contains four nodes n_0, n_1, n_2 and n_3. Each node n_i is connected to the node $n_{(i+1) \bmod 4}$ by two unidirectional channels, named a_i and b_i. The routing function is defined by:

—a message arrived at its destination is evacuated.

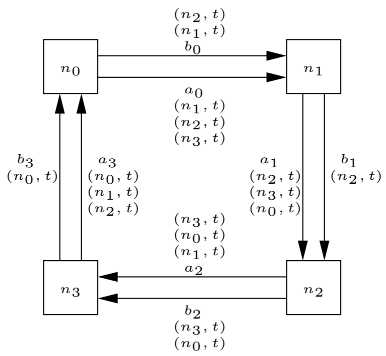


Fig. 7. The interconnect with all edges labeled with messages transiting through channels

—any message can be send through a channel a_i.
—on channel b_i, a X-first routing is used.

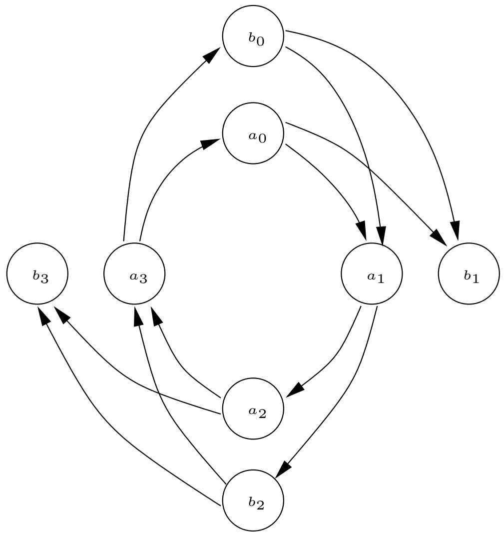


Fig. 8. The EDG of the network of Figure 7

After having checked for connectedness, the EDG is built following algorithm of step 2. Thus, for each channel of the interconnect we find all its dependencies and add them to the EDG. The EDG is shown on Figure 8. This EDG contains cycles. We extract all the SCC of this EDG using the algorithm of Tarjan. There is only one SCC containing channels a_0, a_1, a_2, a_3, b_0 and b_2.

Now, to prove this network deadlock-free, we have to find a connected routing subfunction with an acyclic EDG. To find such a subfunction we try to remove some dependency in the EDG. This is done in step 4. Assuming we start with the node a_0: the labels of this node are (n_1, t), (n_2, t) and (n_3, t). Messages destined for the node n_1 have reached their destination. Messages destined for the node n_2

can be sent though channel b_1 which is not in the SCC. Finally, messages destined for the node n_3 can not be sent through a channel not in the SCC, the dependency between a_0 and a_1 cannot be removed, hence a_1 cannot be removed from the SCC.

Now, we consider the node b_0: the labels associated with this node are (n_1, t) and (n_2, t). Messages of label (n_1, t) have reached their destination. Messages of label (n_2, t) can be send through the channel b_1, which is not in the SCC. Hence, we can restrict the routing function by not allowing the node n_1, to send messages (n_2, t) through the channel a_1. Thus, there is no dependency between b_0 and a_1 anymore. b_0 can be discarded from the SCC.

Now we consider the node a_3. The labels of this node are (n_0, t), (n_1, t) and (n_2, t). Messages destined for the node n_0 have reached their destination. Messages of labels (n_1, t) and (n_2, t) can be sent through the channel b_0, hence a_3 can be emptied by forwarding message to b_0. Thus, we can restrict the routing subfunction to not supply a_0 as next channel to messages (n_1, t) and (n_2, t) in the node n_0. Thus, there is no dependency between a_3 and a_0 anymore and a_3 can also be removed from the SCC. Thus the SCC is broken. Now we have found a routing subfunction with an acyclic EDG, thus this network is deadlock free.

5. EXPERIMENTATION AND COMPARISON WITH STANDARD APPROACH

This section presents the standard approach to prove the deadlock-freeness of a network. This standard approach is the reference we compare our algorithm with. Then a set of benchmarks is presented, and finally the performance of the standard approach and of our algorithm are given and discussed.

5.1 The standard approach

Here we present a alternative algorithm to check if a network is deadlock free. It is an adaptation of [Schwiebert and Jayasimha 1996]. It consists in finding all cycles in the EDG and it tries to break each of them:

(1) Build the EDG associated with the routing function.
(2) Find a cycle in the EDG.
(3) Restrict the routing function such that one edge of the cycle is removed.
(4) Check whether the interconnect remains connected.
(5) If the interconnect does not remain connected, go to step 3 and process with the next edge of the cycle
(6) Construct the new EDG
(7) Go to the step 2 until there is no more cycles in the EDG.

Finding all cycles in an directed graph may take exponential time. When restricting the routing subfunction, new cross dependencies can be created in the EDG of the new routing subfunction. So, each time the routing subfunction is restricted, we have to rebuild the EDG, and then find all cycles of this new EDG.

Here is the detailed algorithm adapted from [Schwiebert and Jayasimha 1996]:

INPUTS: the EDG C
Find all cycles in C
for each cycle c of C **do**

```
  for each  edge v of c do
    Remove v from C
    if the interconnect remain connected then
      Construct the new EDG
      apply this algorithm on the new EDG
      if all cycles has been broken then
         return YES // the network is deadlock free
      endif
    else
      process with the next edge
    endif
  endfor
endfor
return NO // the network is not deadlock free
```

5.2 Benchmarks

We compared the standard algorithm with the one described in Section 4 on a benchmark suite representative of NoC as described in Section 2.1 composed of the following network topologies combined with different routing functions; these topologies are common topologies for network on chips with or without virtual channels. The number of nodes used in the benchmark suite is comparable to those find in current NoC. Some of them are deadlock-free and others possibly lead to deadlocks.

Two scalable topologies are considered: a $2D$ mesh as in DSPIN [Panades et al. 2006] and a *fat-tree* as in SPIN [Guerrier and Greiner 2000].

The $2D$ **mesh** is a $n \times n$ mesh with two unidirectional channels connecting two neighbor nodes. The following routing functions are considered in case of a unique type of messages:

(1) X-first (XY): a X-first routing is used. Messages are first routed on the X axes, then on the Y axes. This is a deterministic non-adaptive routing function that prevents deadlocks.
(2) Shortest Path (SP): messages follow one of the shortest path on the grid. This is an adaptive routing function that may lead to deadlocks.

In order to define a deadlock-free adaptive routing scheme for the $2D$ mesh, we extend the set of channels of the mesh as follows: a *couple* of channels connects two neighbors nodes in each direction. For each couple of channels, one is selected by shortest path strategy, and the other one is selected by the X-first strategy. This routing scheme is called “Shortest path with escape path (SPEP)”; It is adaptive and deadlock-free.

The Fat Tree topology presented here connects each nodes to its four neighbor’s in the successive layers with one pair of unidirectional channels; there are two types of messages: request and response; messages of type response have the highest priority. Two routing functions are considered:

(1) No Separate Path (NSEP): all channels are shared by request and response messages. The routing function is adaptive when messages going upwards to

Table I. number of node and time to check an interconnect using SCC

messages	I	R	# of nodes	time	deadlock-free
1 type	mesh	XY	4900	1624.80s	yes
		SP	3025	1342.52s	no
		SPEP	256	19.68s	yes
2 types	fat tree	NSEP	256	21.83s	no
		SEP	256	11.29s	yes

Table II. number of node and time to check an interconnect using cycles

messages	I	R	# of nodes	time	deadlock-free
1 type	mesh	XY	3600	1314.80s	yes
		SP	100	6.03s	no
		SPEP	100	21.24s	yes
2 types	fat tree	NSEP	256	43.40s	no
		SEP	256	21.94s	yes

the roots, and non-adaptive for messages going downwards to the leafs. It may lead to deadlocks.

(2) Separate path (SEP): request and response messages follow separate paths. Paths are split into disjoint sets, one dedicated to request and one dedicated to response. The routing function is adaptive when messages going upwards to the roots, and non-adaptive for messages going downwards to the leafs. It is deadlock-free.

All the experiments have been performed on a 3GHz pentium4 workstation with 512Mo of RAM. These are summed up in Table I.

Table I presents three informations: the largest topology processed by the tool (measured in number of nodes), the computation time expressed in seconds and the result of the analysis: the topology associated with the routing function is deadlock-free (yes) or may not be deadlock-free (no).

First of all, the experiment demonstrates that it is possible in practice to see at early stage of the design if a network topology combined with a routing function is guarantied to be deadlock-free or not. This information is of great help for designers who can convince themselves that an architectural choice is deadlock-free.

This analysis can be performed on complex networks and with non-trivial routing functions: with a simple routing function, a network of 4900 nodes can be processed. For more complex routing functions, network with 256 nodes can be processed. Even with complex routing function the computation time remains small: only 20 seconds where necessary to check a mesh with 256 nodes with routing function Shortest Path with Escape Path (SPEP).

In Table II, we can see that the time need to find and break all cycles are much more important than the time needed to find and break CFC. Indeed, finding all cycles are done in exponential time, while finding all SCC can be done in polynomial time. Then, when breaking cycles we need to check if the interconnect remain connected, our condition on breaking SCC ensures that the interconnect remain connected. The standard approach cannot solve meshes greater than 10×10.

Tables III and V present the computation time in seconds of different steps of our algorithm when the number of nodes increases while keeping the same routing scheme. Considered steps are:

Table III. detail of computation time for a mesh with SPEP routing using SCC

# of nodes	Labeling edges	Building EDG	Breaking SCC	Total
100	0.04	0.06	0.70	0.80
121	0.06	0.11	1.37	1.54
144	0.09	0.15	2.39	2.63
169	0.15	0.25	4.17	4.57
256	0.49	0.73	18.44	19.68

Table IV. detail of computation time for a mesh with SPEP routing using cycles

# of nodes	Labeling edges	Building EDG	Breaking cycles	Total
100	0.03	0.05	14.54	14.62
121	0.06	0.09	30.45	30.60
144	0.09	0.15	58.03	58.27

Table V. detail of computation time for a fat tree with NSEP routing using SCC

# of nodes	Labeling edges	Building EDG	Breaking SCC	Total
4	0.00	0.00	0.00	0.00
8	0.00	0.01	0.00	0.01
32	0.18	0.19	0.00	0.37
256	9.46	12.14	0.01	21.83

—Labeling edges of I with respect to R

—Building the EDG for each R_t

—Breaking all SCC for each R_t

Tables IV and VI present the computation time of different steps using cycles when the number of nodes increases while keeping the same routing scheme.

While breaking the SCC takes a lot of time for the mesh with routing function SPEP (Table III), labeling the edges takes most of the time for the fat tree NSEP (Table V). There are more channels for the fat tree than for the mesh. In the other hand, checking deadlock-freeness for the fat tree NSEP is trivial, since there is no separate path for request and response, so $R_{response}$ contains no channel and is not connected. So we can not guaranty this network to be deadlock-free, and in fact it contains deadlocks [Charlery et al. 2004].

In Tables III and IV, we can see that while the time to break SCC take few seconds for mesh up to 169 nodes, breaking cycle take about 1 minute for a mesh of 144 nodes.

In Tables V and VI, we can see a huge difference between the time needed to break SCCs and cycles, while the EDG of this routing function is deadlock-free. This overhead is due to the complexity of the algorithm described in section 5.1, especially the necessity to check the connectedness of the routing function each time an edge of the EDG is removed, and to rebuild EDGs. This overhead becomes critical as the number of channel grows.

These two networks (NSEP and SEP) present a favorable configuration for the standard approach since NSEP has no cycle in its EDG, and when breaking a cycle in the EDG of SPEP, the network remains connected, so no bad choice can be made when breaking cycle, which eliminates the backtracking phases which may be very costly.

We have presented our results on a benchmark suite of NoC of hundred of nodes.

Table VI. detail of computation time for a fat tree with NSEP routing using cycles

# of nodes	Labeling edges	Building EDG	Breaking cycles	Total
4	0.00	0.00	0.00	0.00
16	0.00	0.00	0.01	0.01
32	0.19	0.18	0.36	0.73
256	9.42	12.58	21.21	43.40

Deadlock-freeness have been checked for classical topologies with realistic size and with different routing functions. The short time needed to find deadlock or to prove the deadlock-freeness of a NoC of hundred of nodes validates this approach for production class NoC.

6. CONCLUSION

Taking into account the environment is mandatory to check whether a SoC is deadlock-free. A first contribution of this paper is an extension to Duato's theory that takes into account the environment of a network on chip by ordering different types of messages. With our approach, protocol and interconnect is combined in a unified way. A current trend in NoC design is to provide customisable NoC which allow application-specific design of NoC [Stergiou et al. 2005],[Goossens et al. 2005] which permit such a co-design. This approach allows a partial sharing of channels by different types of messages. Hence, it improves the resource usage and, in some cases, decrease resource needs compared to a standard duplication approach.

A second contribution is a new algorithm to determine whether an interconnect combined with a routing function is deadlock-free. Our approach is based on the analysis of "Strongly Connected Components" (SCC) of the Extended Dependency Graph of the routing function, instead of classical "cycle" analysis. For this purpose, we proposed a sufficient condition to make SCC reducible. These contributions leads to the development of a prototype tool (ODI). A third contribution is an experimentation of this tool on a set of typical NoCs and a faithful comparison with the standard cycle reduction algorithm of [Schwiebert and Jayasimha 1996]. We have shown that our tool can analyse networks of hundred of nodes, and in a shorter time than the classical algorithm. Furthermore, false deadlock were never detected. Despite our condition is not necessary, it is not too coarse and it captures efficiently real deadlocks.

This tool may be incorporated in the design of NoC process at early stages: it provides an easy way to verify that a routing function for an interconnect is deadlock-free. Thus it allows designers to define more sophisticated routing functions which are better adapted to their needs (because they limit the hardware introduced to avoid deadlocks). We hope this tool will contribute to develop new adaptive strategies for NoC. In fact, since regular topologies cannot be guarantied anymore by the fabrication process, more sophisticated routing function should be considered to deal with irregularities that can appear in the interconnect due to defective nodes.

Further developments involve several directions: 1) Although all deadlocks suspected by the tool correspond to real deadlocks, we are interested in finding a *necessary and sufficient* condition to suppress SCC. Further works will also include the extension of this theory in order to support a larger class of routing function, especially source routing functions. 2) Another issue to address is parameterizable networks (a proof independent of the size of the net). This issue needs the definition of a symbolic representation of the dependency graph and the adaptation (or definition of news) algorithm to break SCC.

REFERENCES

Bjerregaard, T. and Sparso, J. 2005. A Router Architecture for Connection-Oriented Service Guarantees in the MANGO Clockless Network-on-Chip. In *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*. IEEE Computer Society, Washington, DC, USA, 1226–1231.

Bolotin, E., Cidon, I., Ginosar, R., and Kolodny, A. 2004. QNoC: QoS architecture and design process for network on chip. *J. Syst. Archit. 50,* 2-3, 105–128.

Bolotin, E., Cidon, I., Ginosar, R., and Kolodny, A. 2007. Routing Table Minimization for Irregular Mesh NoCs. In *DATE '07: Proceedings of the conference on Design, Automation and Test in Europe*.

Charlery, H., Greiner, A., Encrenaz, E., Mortiez, L., and Andriahantenaina, A. 2004. Using VCI in a on-chip system around SPIN network. In *11th International Conference Mixed Design of Integrated Circuits and Systems*. 571–576.

Chien, A. A. 1998. A Cost and Speed Model for k-ary n-Cube Wormhole Routers. *IEEE Trans. Parallel Distrib. Syst. 9,* 2, 150–162.

Dally, W. J. and Seitz, C. L. 1987. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Trans. Comput. 36,* 5, 547–553.

Dally, W. J. and Towles, B. 2001. Route Packets, Not Wires: On-Chip Interconnection Networks. In *DAC*. ACM, 684–689.

Duato, J. 1991. On the Design of Deadlock-Free Adaptive Routing Algorithms for Multicomputers: Design Methodologies. *505*, 390–405.

Duato, J. 1995. A Necessary and Sufficient Condition for Deadlock-Free Adaptive Routing in Wormhole Networks. *IEEE Trans. Parallel Distrib. Syst. 6,* 10, 1055–1067.

Fleury, E. and Fraigniaud, P. 1998. A General Theory for Deadlock Avoidance in Wormhole-Routed Networks. *IEEE Trans. Parallel Distrib. Syst. 9,* 7, 626–638.

Goossens, K., Dielissen, J., Gangwal, O. P., Pestana, S. G., Radulescu, A., and Rijpkema, E. 2005. A Design Flow for Application-Specific Networks on Chip with Guaranteed Performance to Accelerate SOC Design and Verification. In *DATE*. IEEE Computer Society, 1182–1187.

Goossens, K., Dielissen, J., and Radulescu, A. 2005. AEthereal Network on Chip: Concepts, Architectures, and Implementations. *IEEE Design & Test of Computers 22,* 5, 414–421.

Guerrier, P. and Greiner, A. 2000. A Generic Architecture for On-Chip Packet-Switched Interconnections. In *DATE*. IEEE Computer Society, 250–256.

Ianni, M. D. 1997. Wormhole Deadlock Prediction. In *Euro-Par*, C. Lengauer, M. Griebl, and S. Gorlatch, Eds. Lecture Notes in Computer Science, vol. 1300. Springer, 188–195.

Jayasimha, D. N., Schwiebert, L., Manivannan, D., and May, J. A. 2003. A foundation for designing deadlock-free routing algorithms in wormhole networks. *J. ACM 50,* 2, 250–275.

Kim, J., Park, D., Theocharides, T., Vijaykrishnan, N., and Das, C. R. 2005. A low latency router supporting adaptivity for on-chip interconnects. In *DAC '05: Proceedings of the 42nd annual conference on Design automation*. ACM Press, New York, NY, USA, 559–564.

Knuth, D. E. 1993. *The Stanford GraphBase : a platform for combinatorial computing.*

Lin, X., McKinley, P. K., and Ni, L. M. 1995. The Message Flow Model for Routing in Wormhole-Routed Networks. *IEEE Transactions on Parallel and Distributed Systems 6,* 7, 755–760.

Ni, L. M. and McKinley, P. K. 1993. A Survey of Wormhole Routing Techniques in Direct Networks. *Computer 26,* 2, 62–76.

Panades, I. M., Greiner, A., and Sheibanyrad, A. 2006. A Low Cost Network-on-Chip with Guaranteed Service Well Suited to the GALS Approach. *NanoNet*.

Schwiebert, L. and Jayasimha, D. N. 1996. A Necessary and Sufficient Condition for Deadlock-Free Wormhole Routing. *Journal of Parallel and Distributed Computing 32,* 1, 103–117.

Stergiou, S., Angiolini, F., Carta, S., Raffo, L., Bertozzi, D., and Micheli, G. D. 2005. ×pipes Lite: A Synthesis Oriented Design Library For Networks on Chips. In *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*. IEEE Computer Society, Washington, DC, USA, 1188–1193.