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

Sami Taktak
LIP6 Lab, University Paris VI, CNRS
4, place Jussieu
75252 Paris cedex 05, France
Telephone: +33 1 44 27 70 16
Fax: +33 1 44 27 72 80
sami.taktak@lip6.fr

Emmanuelle Encrenaz
LSV – ENS Cachan & CNRS – France
emmanuelle.encrenaz@lsv.ens-cachan.fr

Jean-Lou Desbarbieux
LIP6 Lab, University Paris VI, CNRS
jean-lou.desbarbieux@lip6.fr

***Abstract*— 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.**

I. 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 [1]. 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 [2] a review of wormhole routing techniques.

Deadlock is an important problem for a network on chip design. In practice, network designers are used to duplicate hardware to avoid resources 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 [3] 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 [4] 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 [5].

Fraigniaud [6] 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: vertex-dependent, input-dependent, source-dependent, history-dependent, path-dependent, multi-dependent, library-dependent, compact routing.

Schwiebert and Jayasimha [7] 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 [8], Schwiebert and Jayasimha have extended their theory to support a larger class of routing functions.

All these theories assume that a message arriving at its destination is eventually consumed. As example, in the SPIN/VCI [9] 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 [3], [4], [6], [7], [8]. In this paper we propose an adaptation of Duato's theory to take into account the dependencies introduced by the environment of the network.

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 [7], [10], [4], [11], [12]. 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 connexity of the routing function. *This technique avoids to check for connexity*. 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.

The remainder of the paper is organized as follows. Section II presents an extension of Duato's necessary and sufficient condition to represent the dependencies between injection and delivery channels due to the environment. Then it presents a sufficient condition to eliminate a cycle that is not involved in a deadlock. Section III presents an original algorithm based on this condition to determine automatically if a network is deadlock-free. Section IV presents the application of our tool on a set of real networks on chip with different routing strategies to detect potential deadlock, and discusses the results. Then we conclude and sketch future directions of work.

II. Theoretical aspect

A. 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 consume 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 [4]. 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 can 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 of 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.

B. Definitions

This section defines precisely the network topology and routing function. The definitions are mostly taken from [4] 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 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. The node source (resp. destination) of a channel c is named $s(c)$ (resp. $d(c)$).

Definition 2: $F = \{\text{free}, \text{busy}\}$ is the *set of valid states* of a channel.

Definition 3: 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 4: A *message* is represented as a pair in $N \times T$ defining its destination and its type.

Definition 5: $label(c)$ is a set of pairs in $N \times T$, each of whom represents a message that can be sent through the channel c.

Definition 6: 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 7: 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 8: 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 9: 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 10: 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 11: A *valid configuration* is a configuration that can be reach from an empty network which is filled with respect to the routing function.

Definition 12: A *deadlock configuration* is a nonempty configuration where no message can progress.

Definition 13: A routing function R for an interconnect I is deadlock-free iff there is no valid deadlock configuration for this routing function.

Definition 14: 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 $c_i, c_j \in C$:

- 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

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

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

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

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

  There is a 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

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

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

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

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

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

  There is a 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.

Definition 15: A *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 1 (Duato's theorem [4]): 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 16: 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

$$\forall c \in R_t(x, y), c \in R(x, y, t) \text{ and } c \notin R(x, y, t')$$

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: 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.

■

C. Construction of the routing subfunction

Given a routing function R, an interconnect I and an ordered set of messages types, to prove R to be deadlock-free, we seek to find 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 cycles of the EDG as suggested in [7]. 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 connexity 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 the 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 be c_0 a channel of the cycle and let be c_k the latest channel on the cycle that can receive a message from c_0. Then we look if any message that came from c_0, in channels c_0 to c_k can be forwarded to channels not belonging to the cycle. Thus such 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 by not allowing any message transiting through c_0 to be transmit to any c_i $(i = 1, \ldots, k)$. R_1 will not have this cycle in it's EDG.

Condition 1 (A sufficient condition to suppress cycles): If, for a routing function R and an interconnect I, there is a cycle such as:

$\exists\, c_0, \ldots, c_k \in cycle$, such as

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

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

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

then this cycle will not lead to a deadlock configuration.

Sketch of the proof:

Let us consider a cycle C made of channels $c_i, i = 0, \ldots, k$ which satisfy condition 1. Let us suppose there is a deadlock configuration for this cycle. Since there is a deadlock in C, c_0 is not empty and contains a message m of type t. There are two cases :

- c_0 contains the head of m. Since C satisfies condition 1, $\exists\, y_0$ such as $y_0 \in R(d(c_0), dest(m), t)$. That is, $d(c)$ can forward m to y_0 and there is no deadlock.
- c_0 does not contain the head of m, $\exists\ c_j$ such as c_j contain the head of m, and $\exists\ y_j$ such as $y_j \in$

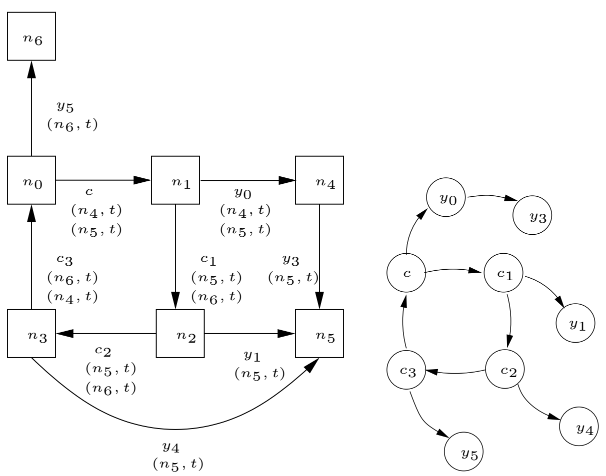
$R(d(c_j), dest(m), t)$. That is, m can be forwarded to y_j, and there is no deadlock.

Thus, there cannot be a deadlock in C.

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 send 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 1(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 1(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



(a) The interconnect and a routing function R_t

(b) The extended dependency graph

Fig. 1. An example of cycle where a deadlock cannot occur

- 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

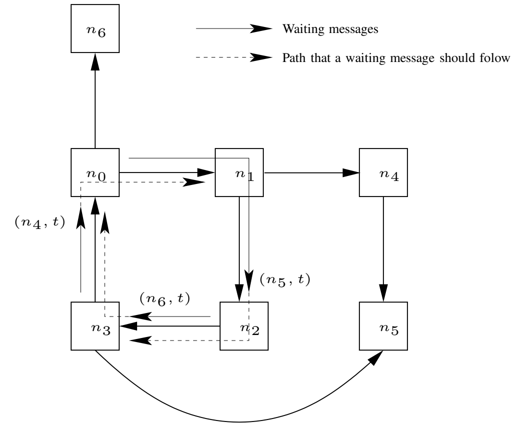


Fig. 2. A deadlock configuration

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 2). Such a cycle does not satisfy condition 1, hence it cannot be suppressed. The SCC is not reducible, the network is not guaranteed to be deadlock-free.

III. Automatic Proof of deadlock free network

In this section, we describe a new algorithm based on theorem 2 and condition 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 has follows:

- split R in subfunctions R_t according to T
- for each t in T:
  1) check for connexity of R_t
  2) construct the EDG of R_t
  3) find all SCC of R_t
  4) break each SCC using condition 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 the dependency of each arcs.

A. Construction of $EDG(R_t)$

```
for each type T:
 for each arc a:
   add a to the dependency graph
 for each node n of the dependency graph:
```

```
for each outgoing arc a of n:
    add each direct dependency of a to the EDG(R_t)
    add each indirect dependency of a to the EDG(R_t)
```

B. Finding SCC in $EDG(R_t)$

We use the well known algorithm of Tarjan (ref) that returns the set of SCC of $EDG(R_t)$.

C. 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.

```
for all strong component SCC
    if it's not reduced to one node
        call:Suppress_Cycle(SCC)
```

1) suppress cycle:

```
Suppress_Cycle(C):
for all nodes v of the C
    if call:Exists_Escape_Path (v, labels(v))
        remove v from the C
```

2) escape path:

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

    let nlechap = {(d,t)|(d,t) ∈ l and ∀u ∉ C, d(v) ≠ s(u)}
//lnechap is the subset of l whose message cannot be
//evacuated through an escape node leaving the current SCC
C.

    l ← l \ {(d,t)|d = v}
//remove labels of messages having reached their destination
    if nlechap ≠ ∅ a deadlock is possible -> return NO
    else forall u ∈ C such that d(v) = s(u):
      if call:Exists_Escape_Path(u, l ∩ labels(u)) = NO
          return NO

    return YES
```

D. 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 the figure 3 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.
- any message can be send through a channel a_i.
- on channel b_i, a X-first routing is used.

The first step is to build the EDG. It is built using the algorithm described in section III-A. Thus, for each channel of the interconnect we find all its dependencies and add them

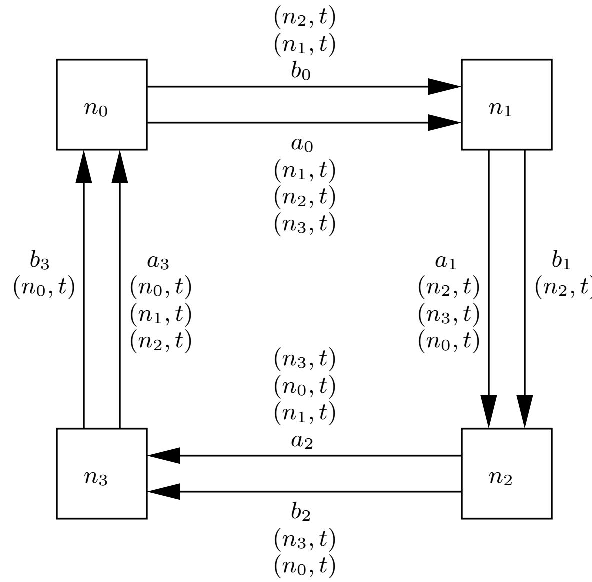


Fig. 3. The interconnect with all edges labeled with messages transiting through channels

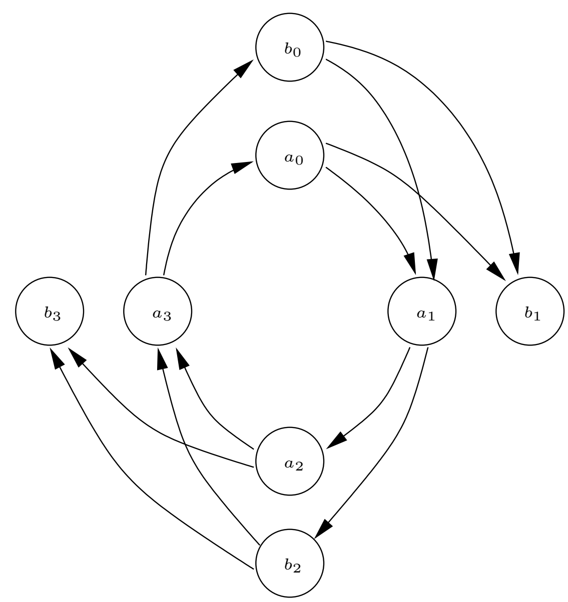


Fig. 4. The EDG of the network of the figure 3

to the EDG. The EDG is shown on figure 4. This EDG contains cycles. We extract all the SCC of this EDG using the algorithm of Tarjan presented in III-B. 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 as described in section III-C. 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 send 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.

IV. Experimental Result

We implemented an automatic method to check if a network is deadlock-free, based on theorem 2 and condition 1 presented in the previous section. The current section presents the application of this tool to a set of network topologies combined with different routing functions, some of them being deadlock-free and others possibly leading to deadlocks.

Two scalable topologies are considered: a $2D$ mesh as in DSPIN [13] and a *fat-tree* as in SPIN [14].

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 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. They are summed up in Table I.

Interconnect	routing function	# of nodes	time	deadlock-free
mesh	XY	4900	1624.80s	yes
	SP	3025	1342.52s	no
	SPEP	256	19.68s	yes
fat tree	NSEP	256	11.29s	no
	SEP	256	21.83s	yes

TABLE I

NUMBER OF NODE AND TIME TO CHECK AN INTERCONNECT

Table I presents three information: 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 fact, computing time seems not to be the limitation. Actually, limitation comes from memory occupation

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

- Labeling edges of I with respect to R
- Building the EDG for each R_t
- Breaking all SCC for each R_t

Times are given in seconds.

While breaking the SCC takes a lot of time for the mesh with routing function SPEP, labeling the edges takes most of the time for the fat tree NSEP. There are more channels for the fat tree than for the mesh. In the other hand, checking

# of nodes	Labeling edges	Building EDG	Breaking SCC	Total
4	0.00	0.00	0.00	0.00
25	0.00	0.00	0.00	0.00
100	0.04	0.06	0.70	0.80
256	0.49	0.73	18.44	19.68

TABLE II

DETAIL OF COMPUTATION TIME FOR A MESH WITH SPEP ROUTING

# 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

TABLE III

DETAIL OF COMPUTATION TIME FOR A FAT TREE WITH NSEP ROUTING

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 [9].

V. CONCLUSION

We have extended Duato's theory to take into account the environment of a network on chip by ordering different types of messages. Then we proposed a new algorithm to determine if an interconnect combined with a routing function is deadlock-free using Duato's theorem. This 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. Then we proposed a sufficient condition to make such a SCC reducible. Finally, we presented some results obtained with our tool that shows the effectivness of this approach: networks of hundreds of nodes are analysed very quickly.

This tool can 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.

Although all deadlocks suspected by the tool corresponded to real deadlocks, we are interested in finding a *necessary and sufficient* condition to suppress SCC. Futher works will also include the extension of this theory in order to support a larger class of routing function, especially "store and forward" routing functions.

REFERENCES

[1] W. J. Dally and B. Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks." in *DAC*. ACM, 2001, pp. 684–689.

[2] L. M. Ni and P. K. McKinley, "A Survey of Wormhole Routing Techniques in Direct Networks," *Computer*, vol. 26, no. 2, pp. 62–76, 1993.

[3] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Trans. Comput.*, vol. 36, no. 5, pp. 547–553, 1987.

[4] J. Duato, "A Necessary and Sufficient Condition for Deadlock-Free Adaptive Routing in Wormhole Networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 6, no. 10, pp. 1055–1067, 1995.

[5] A. A. Chien, "A Cost and Speed Model for k-ary n-Cube Wormhole Routers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 9, no. 2, pp. 150–162, 1998.

[6] E. Fleury and P. Fraigniaud, "A General Theory for Deadlock Avoidance in Wormhole-Routed Networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 9, no. 7, pp. 626–638, 1998.

[7] L. Schwiebert and D. N. Jayasimha, "A Necessary and Sufficient Condition for Deadlock-Free Wormhole Routing," *Journal of Parallel and Distributed Computing*, vol. 32, no. 1, pp. 103–117, 1996.

[8] D. N. Jayasimha, L. Schwiebert, D. Manivannan, and J. A. May, "A foundation for designing deadlock-free routing algorithms in wormhole networks," *J. ACM*, vol. 50, no. 2, pp. 250–275, 2003.

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

[10] J. Duato, "On the Design of Deadlock-Free Adaptive Routing Algorithms for Multicomputers: Design Methodologies." vol. 505, pp. 390–405, 1991.

[11] X. Lin, P. K. McKinley, and L. M. Ni, "The Message Flow Model for Routing in Wormhole-Routed Networks." *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, no. 7, pp. 755–760, 1995.

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

[13] "DSPIN." [Online]. Available: http://www.lip6.fr/Direction/2005-05-13-DSPIN.pdf

[14] P. Guerrier and A. Greiner, "A Generic Architecture for On-Chip Packet-Switched Interconnections." in *DATE*. IEEE Computer Society, 2000, pp. 250–256.