Computers and Electrical Engineering xxx (2014) xxx–xxx

Contents lists available at ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

Distributed topology discovery in self-assembled nano network-on-chip ☆

Vincenzo Catania, Andrea Mineo, Salvatore Monteleone, Davide Patti *

DIEEI, University of Catania, V.le Andrea Doria 6, 95125 Catania, Italy

ARTICLE INFO

Article history:
Received 24 February 2014
Received in revised form 12 September 2014
Accepted 12 September 2014
Available online xxxx

Keywords:
Nanotechnology
DNA
Self-assembly
Routing
Deadlock

ABSTRACT

In this paper, we present DiSR, a distributed approach to topology discovery and defect mapping in a self-assembled nano network-on-chip. The main aim is to achieve the already-proven properties of segment-based deadlock freedom requiring neither a topology graph as input, nor a centralized algorithm to configure network paths. After introducing the conceptual elements and the execution model of DiSR, we show how the open-source Nanoxim platform has been used to evaluate the proposed approach in the process of discovering irregular network topology while establishing network segments. Comparison against a tree-based approach shows how DiSR still preserves some important properties (coverage, defect tolerance, scalability) while avoiding resource hungry solutions such as virtual channels and hardware redundancy. Finally, we propose a gate-level hardware implementation of the required control logic and storage for DiSR, demonstrating a relatively acceptable impact ranging from 10 to about 20% of the budget of transistors available for each node.

1. Introduction and motivation

Exploring long-term alternatives to the CMOS technology is gaining more and more relevance as the scaling trend of such devices keeps introducing new challenges: power density, defect tolerance, testing costs and wire delays are only a few of the many critical aspects involved [1]. While software parallelism and multicore approaches [2,3] are partially mitigating the impact of such constraints on performances, it is likely that a growing computing demand will eventually need even more radical architectural modifications and paradigm shifts to address the Computer Design challenges of the upcoming decades.

In recent years, self-assembled nanoscale architectures [4] emerged as a promising technology due their tera/peta scale of integration, defect tolerance and huge potential computing capabilities. These technologies are certainly still at their early stage of development; however, different laboratory demos and proofs-of-concept have been presented [5,6]. For a complete survey on self-assembled architectures and how they compare against classical many-core systems in terms of technologies, fault tolerance, performance and software tools see also [7]. The main idea behind this approach is exploiting the physical regularity and stability of DNA structures in order to create a scaffold onto which nano-devices (e.g. nanowires and CNFETs [8,9]) can be attached. This can be achieved by designing appropriate complementary DNA tags for each terminal to be

☆ Reviews processed and recommended for publication to the Editor-in-Chief by Guest Editor Dr. Masoud Daneshtalab.

* Corresponding author.
E-mail addresses: vincenzo.catania@dieei.unict.it (V. Catania), andrea.mineo@dieei.unict.it (A. Mineo), salvatore.monteleone@dieei.unict.it (S. Monteleone), davide.patti@dieei.unict.it (D. Patti).

http://dx.doi.org/10.1016/j.compeleceng.2014.09.003
0045-7906/

placed, so that a nano device will be attached only where its own DNA tag matches a complementary tag on the DNA grid scaffold (see Fig. 1a). Of course, a detailed description of the chemical properties involved is far beyond the scope of this paper (see also [10]).

In this work, we present DiSR (Distributed Segment Routing), a distributed approach focused on addressing the three main challenges that this new fabrication process introduces in nanoscale Network-on-Chip Design: *(i) limited node complexity*, *(ii) large scale randomness* and *(iii) high defect rates*. The *limited node complexity* aspect is directly related to the use of complementary DNA tags in order to place circuit components. The traditional CMOS process introduces complexity with larger photolithography masks: more complex (larger) circuits will only require larger masks. Conversely, a self-assembly process achieves complexity by increasing the number of unique DNA tags, since having more different tags means having more control on component placement. Ideally, by specifying a single and unique tag for each nano device terminal, we could exactly choose where each component would be placed in the design, but the number of DNA symbols forming the DNA sequence is limited (sequence of 4 nucleobases G, A, T, C) and so creating many different tags (of a predetermined length) would mean making them more similar to each other, increasing the probability of incorrect/partial matching. To avoid this problem, the number of unique tags must be limited, which limits also the complexity available at each node. In this work, a budget of about 10,000 CNFETs per node has been assumed, as estimated in [11]. *Large scale randomness* is the other fundamental condition of self-assembled technology: DNA tags allow controlled placement inside the node grid, but there is no control over the placement of these grids in the whole network. As a consequence, other typical properties of regular networks cannot be guaranteed, e.g. being connected to a fixed number of neighbors, having a determined orientation and so on. Finally, *defect rates*: while they are hardly tolerated in mask-based top-down design, the same nature of a bottom-up self-assembly process cannot assume such a deterministic device placement process, thus defect tolerance is more a design requirement than an exception to be avoided.

These aspects of a DNA self-assembly process lead to some important implications to be addressed when approaching to the design of a nanoscale Network-on-Chip architecture: computational model should be based on a distributed network of small processing and storage nodes, randomly placed and interconnected (see Fig. 1b). Proof-of-concept of such architectures can be found for example in [5], where instruction and data operands are moved around the network in order to be processed. In addition, independently of the particular policy chosen to route instruction and data packets, such networks must be able to guarantee deadlock freedom without any prior assumption about the topology.

In this work, we introduce DiSR, a Distributed Segment-based approach to topology discovery and deadlock freedom in large scale DNA self-assembled on chip networks. Our contribution aims to achieve the classical properties of a segment-based approach [12] without requiring any topology graph, external defect map or centralized algorithm execution. The DiSR approach is not intended to discover the "optimal" segment choice (ideally reachable with the knowledge of the topology graph) but just to demonstrate a concrete model that can fit into such complex, irregular and large sized networks. It is important to underline that, although we can assume as background an architecture similar to the one depicted in Fig. 1,

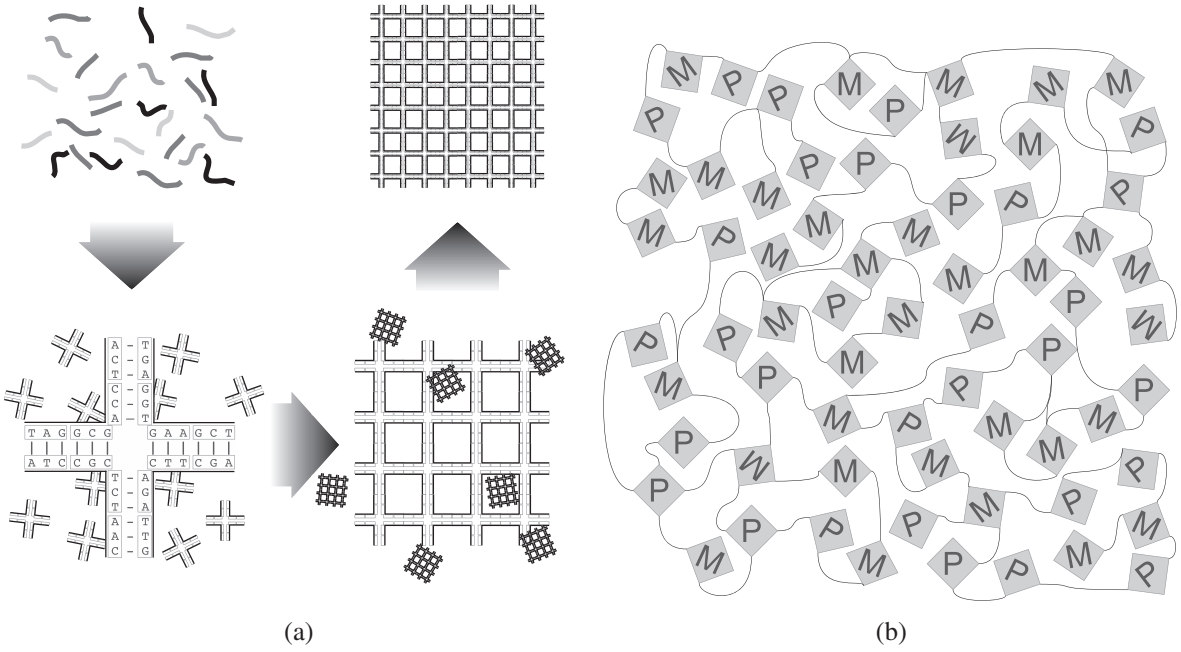

Fig. 1. (a) Strands of complementary DNA tags self assemble to generate larger structures. (b) The resulting irregular topology of processing (P) and storage (M) elements.

Please cite this article in press as: Catania V et al. Distributed topology discovery in self-assembled nano network-on-chip. Comput Electr Eng (2014), http://dx.doi.org/10.1016/j.compeleceng.2014.09.003

the DiSR topology discovery approach presented this work is more general and not strictly dependent on any underlying computational model adopted.

The paper is organized as follows: in Section 2 we summarize the main approaches for topology agnostic deadlock freedom. Next, in Section 3 are described the main elements and the execution model of the proposed approach. Further details about node behavior in relation to the execution model are provided in Section 4. In Section 5, simulations are carried out with the open-source tool Nanoxim to demonstrate how DiSR preserves some of the main segment-based properties and how compares to the tree-based approaches. Finally a draft of an architectural implementation is shown in Section 6 to evaluate the feasibility of the approach in terms of node complexity.

2. Background and contribution

Several works addressed the problem of topology agnostic deadlock freedom introducing virtual channels [13–15], but while they show interesting performances, the low resources available to prevent deadlocks makes them not suitable for the DNA self-assembled networks we are assuming as scenario. Other topology independent algorithms do not require any virtual channel but exploit the concept of "turn prohibition". In particular, authors of [16] exploit the creation of a spanning tree of the topology, placing bidirectional restrictions to avoid that a packet can traverse a given link in both up and down directions. The hierarchical nature of this approach can lead to an uneven traffic distribution, with many packets traversing upper links (near to the root), but this is quite acceptable in classical wide area networks topologies with a limited number of nodes. Other approaches such as FX [17] mitigate this issue, but the set of turn restrictions is still prefixed strictly depending on the particular tree root selected. Other solutions try to approach the issue of irregular topologies by limiting the number [18,19] or the location [20,21] of missing links. This restriction is clearly unacceptable given the hight-defect rates of DNA self-assembled networks scenario. For the same reason, we also avoid considering solutions based on hardware-redundancy to dynamically recover defects as in [22].

In [12] authors present an approach that solves these limitations setting turn restrictions locally, independently from other restrictions. The whole network is partitioned into segments, and a bidirectional turn restriction can be freely chosen within a segment in order to guarantee deadlock freedom while preserving network connectivity. This *locality independence* property, removing the requirement of choosing a particular tree/root, would make this approach the best choice for the given scenario; however, it still requires the knowledge of the whole network graph in order to find the segments.

The DiSR approach presented in this work is a first attempt to achieve the same goals of a segment based turn prohibition requiring neither the knowledge of a topology graph nor centralized execution. In particular, DiSR implements segment discovery and turn prohibition using a completely different mechanism, based on a distributed exchange of small setup packets requiring a limited and scalable portion of the available resource budget. The following main features distinguish DiSR from classic approaches to segment partitioning:

- No centralized entity is globally responsible and/or aware of what is going on, i.e. the status of the DiSR execution is collectively distributed among the nodes.
- No defect map and/or topology graph is used as input, thus, the topology has to be discovered *while* segments are created.
- At the end of the execution, no segment list is created or stored anywhere: each node is only aware of being part of a segment, ignoring the presence of other nodes in the same segment and also the presence of other segments in the network.

3. DiSR overview

The main concept behind the turn prohibition method we propose in this work is the *segment*. A segment S_b is basically a path of consecutive nodes and links, starting with a link attached to a node belonging to a different segment S_a and ending with a link attached to a node of another segment S_c. In other words, a segment is a path connecting two other different segments. The example in Fig. 2 shows the segment with id 1.4, starting with a link connected to the node 9, traversing nodes 4, 3, 2 and ending with the link connected to node 1. Other segments depicted are, for example, 9.2 (link from 9, node 10, link to node 3) and 9.3 (link from 9, node 8, link, node 7, link, node 6, link, node 5, link to node 1). Note also that starting/ending links can coincide and a segment can also be formed by one single link (e.g. the case of 7.2 and 5.4). An exception to the rule that a segment must connect two other segments is the first segment established in the network, called *starting segment* which is a loop beginning/ending on the same node.

A detailed description of the DiSR execution model as depicted in Fig. 4 will be discussed in SubSection 3.3. Roughly speaking, the execution phases consist of:

1. Injection of the DiSR process from an upper layer to set a *bootstrap node*.
2. Broadcasting and confirmation to create the first segment of the subnet.
3. Parallel segment requests starting from assigned nodes to discover other segments.
4. Processing of resulting segment confirmation/cancel packets.
5. Turn prohibition in each confirmed segment.

Please cite this article in press as: Catania V et al. Distributed topology discovery in self-assembled nano network-on-chip. Comput Electr Eng (2014), http://dx.doi.org/10.1016/j.compeleceng.2014.09.003

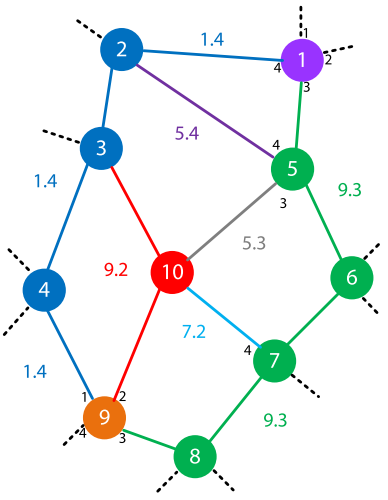

Fig. 2. Example of segments.

This first work is explicitly focused on demonstrating the feasibility and scalability of DiSR in the particular scenario assumed; for this reason we will not investigate the degree of freedom that the last of the above steps introduces, i.e. choosing *where* to place the restriction in the context of a segment. This choice would strictly depend on the particular routing strategy that will be used and could result in some interesting improvements. The scope of this work is understanding how DiSR can implement a distributed topology discovery, mapping defects and creating a segment partitioning that will be exploited in a second design phase which includes choosing and implementing a routing strategy. For this reason, since we can safely assume that a random turn (e.g. the first) can be prohibited once each segment has been confirmed, the last two execution phases will be addressed as one.

3.1. DiSR data structures

We distinct between two different kinds of data stored in each node: *Dynamic Behavior Status* (*DBS*) and *Local Environment Data* (*LED*).

DBS information represent node status and is related to its current execution phase in the context of DiSR. The *DBS* can have the following values:

- *Free*: a node that has not been yet considered by the DiSR algorithm.
- *Bootstrap*: a node which has been explicitly set as bootstrap node from an upper layer via.
- *ActiveSearching*: a node from which a new segments searching process has been started and not yet canceled or confirmed.
- *Candidate*: a node currently candidate to be part of a segment, but not being itself the node from which the searching process was started.
- *CandidateStarting*: same as above, but with the node considered as candidate for starting a segment.
- *Assigned*: a node for which the segment has been determined.

LED variables are a kind of instant snapshot of the execution of DiSR algorithm consisting in the following variables:

- *segID*: a value used to specify the segment to which the node has been assigned or is currently candidate to be part of.
- *visited*: a boolean value. When *true* then a *segID* different from `NULL` specifies the segment to which the node has been assigned.
- *tvisited*: a boolean value. If *true*, the node is being considered as candidate for a segment, and the *segID* value specifies the segment ID for which the node is candidate.
- *link_visited[]*: an array of values containing information about attached links (*segID* of the segment owning each link). When `NULL`, the corresponding link has not yet been assigned.
- *link_tvisited[]*: an array of values representing information about attached links (*segID* of the segment for which the link is candidate). When `NULL`, the link is not candidate for any segment, i.e. it has already been assigned or it is free.

Please cite this article in press as: Catania V et al. Distributed topology discovery in self-assembled nano network-on-chip. Comput Electr Eng (2014), http://dx.doi.org/10.1016/j.compeleceng.2014.09.003

3.2. Message types required

The DiSR approach works with a distributed mechanism which is build upon an exchange of small control packets containing the following fields:

- *packet_type*: encodes the meaning of the control packet.
- *seg_ID*: the id of the segment targeted by the DiSR control message.
- *src_ID*: the id of the node that originated the packet.
- *TTL*: a counter which makes the packet expire after a given number of retries.

With regard to the *packet_type* field, we can have the following control packets types:

- `STARTING_SEGMENT_REQUEST`: Injected by the bootstrap node when searching the first segment.
- `STARTING_SEGMENT_CONFIRM`: used when establishing the starting segment.
- `SEGMENT_REQUEST`: used to search candidates for a segment (not the first).
- `SEGMENT_CONFIRM`: used to establish a segment.
- `SEGMENT_CANCEL`: used to cancel the process of searching a segment along a specific link.

A quantitative analysis of the resources needed to implement and manage these structures is presented in Section 6 where it is discussed the impact of DiSR control logic and storage on the node's hardware implementation.

3.3. Execution model

DiSR control packets described in the previous subsection trigger node events, which change *DBS* according to the current value of *LED* variables and the type of packet received. The main DiSR phases, together with a reference to the particular status transition involved are described in the following. As a convention, we used single letters to refer to one of the *DBS* transitions in Fig. 3 while the phases are visually represented in Fig. 4.

- **Injecting bootstrap request**: all nodes have an initial *DBS* status set to *Free*, except for a node with status *Bootstrap*, set by some control signal from an upper layer via *(a)*. When starting, this bootstrap node changes its status to *CandidateStarting (b)*, and injects a `STARTING_SEGMENT_REQUEST` across one of its free links (Fig. 4a).
- **Bootnode broadcasting**: Each node receiving a `STARTING_SEGMENT_REQUEST`, when *Free*, forwards it to its free links using a flooding mechanism and becoming *CandidateStarting (g)*. Each of its free links is then marked as *tvisited* with the segment id associated to the request. Note that a node that has already received a `STARTING_SEGMENT_REQUEST` packet can simply ignore further packets associated to the same request, having already contributed to the flooding (Fig. 4b).

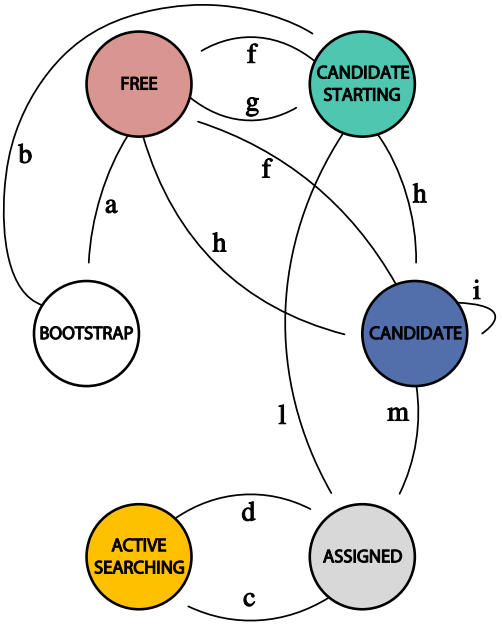

Fig. 3. DiSR node execution model.

Please cite this article in press as: Catania V et al. Distributed topology discovery in self-assembled nano network-on-chip. Comput Electr Eng (2014), http://dx.doi.org/10.1016/j.compeleceng.2014.09.003

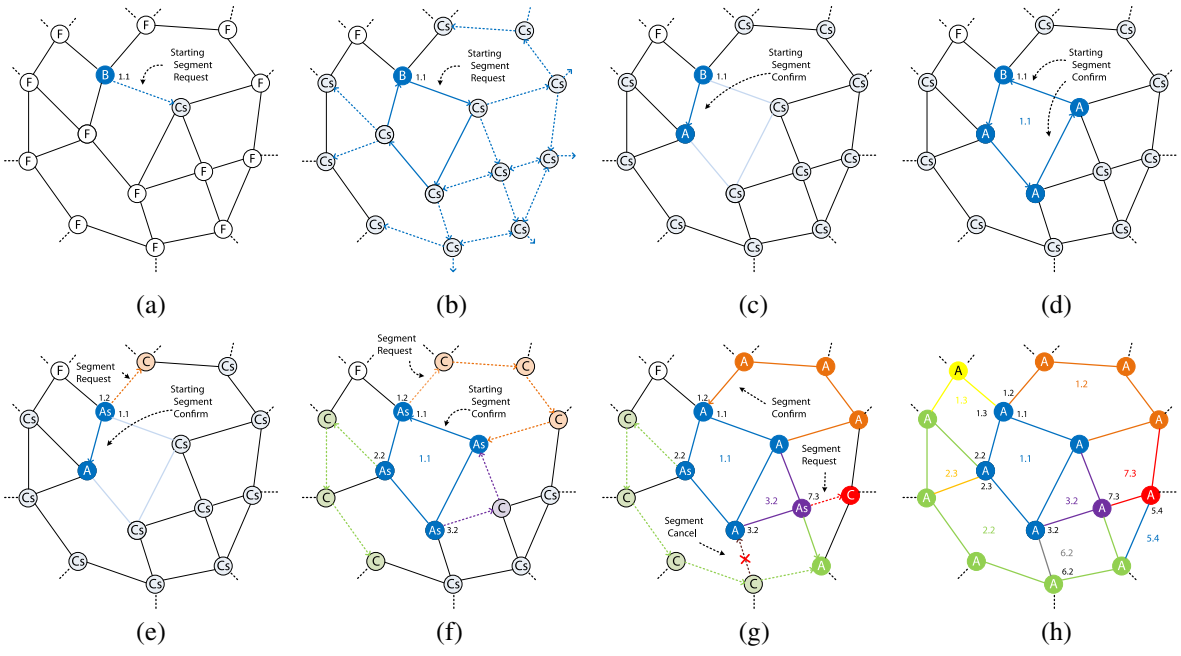

Fig. 4. Example of DiSR segment setup: nodes status is labeled as *CandidateStarting* (*Cs*), *Candidate* (*C*), *Free* (*F*), *Assigned* (*A*) or *Bootstrap* (*B*).

- **Confirming the starting segment**: when a `STARTING_SEGMENT_REQUEST` packet reaches the bootstrap node (from a different link, of course), the starting segment is found. Then the bootstrap node sends back a `STARTING_SEGMENT_CONFIRM` packet along the link from which it received the `STARTING_SEGMENT_REQUEST` and becomes *Assigned (l)*. Each node receiving the confirmation does the same by changing its own status to *Assigned* (Fig. 4c). So the confirmation packet is sent back from node to node and the starting segment is created (Fig. 4d).
- **Injecting segment requests**: Each node in the *Assigned* status can potentially initiate a new segment request. So it check its links and if a free link is found it changes its status to *ActiveSearching (c)* and then sends a `SEGMENT_REQUEST` across the free link *(c)* (see Fig. 4e). Note that each *Assigned* node can inject the request just after becoming *Assigned*, i.e. it is not needed that the whole confirmation process in completed (Fig. 4f). Note also that nodes previously set to *CandidateStarting*, when receive a `SEGMENT_REQUEST` can simply cancel their status and set themselves to *Candidate* for that request, since this new request means that the first starting segment has already been found *(h)*.
- **Segment confirmation**: The segment searching process is successful when an already *Assigned* node receives the `SEGMENT_REQUEST` packet. Then, a `SEGMENT_CONFIRM` is sent back along the path that originated the request (Fig. 4g), while the node remains *Assigned* (since it could confirm other segments). Each node previously set to *Candidate* for that segment id, when receives a confirmation packet changes its status to *Assigned (m)* and send back the same confirmation. When this confirmation reaches the initiator of the request it changes its status from *ActiveSearching* to *Assigned (d)*.
- **Failing while searching a segment**: this happen when a node receives a `SEGMENT_REQUEST` packet but matches one of two the following conditions: the node is *Free* but has no more suitable free links (thus cannot forward the `SEGMENT_REQUEST`)*(f)*; the node is already *Candidate* with another segment id, i.e. is part of another searching process. In all these cases the node sends back a `SEGMENT_CANCEL` along the incoming link (see at the bottom of Fig. 4).

4. Detailed node behavior model

In this Section, we provide further details about DiSR execution model, describing the internal behavior implemented in each node. Note that this also corresponds exactly to what should be implemented from the hardware perspective, as discussed in Section 6.

The *LED* variables (*segID*, *visited*, *tvisited*, *link_visited[]*, *link_tvisited[]*), stored in each node, are initially set to `NULL`, so the node and all its links are considered as *Free*. The only node that makes an exception is the bootstrap node, initialized in the *Bootstrap* status. All the subsequent events happening at each node are consequence of its current dynamic status (DBS), packets received and internal status (represented by the *LED*). Note that in the following, when using expression like "has a free link" or "free node" we will assume the relationship between *LED* and *DBS* discussed in Section 3.1. Furthermore, for each packet type all the cases not explicitly listed can simply be ignored since associated to invalid/inconsistent events of the DiSR execution model. A complete list of these cases is described in the simulator source code available at [23].

Please cite this article in press as: Catania V et al. Distributed topology discovery in self-assembled nano network-on-chip. Comput Electr Eng (2014), http://dx.doi.org/10.1016/j.compeleceng.2014.09.003

4.1. Receiving a STARTING_SEGMENT_REQUEST

The request for the first segment should be managed differently since all nodes (except the bootstrap one) must forward the packet using a broadcasting mechanism. This is necessary since the request packet must return the bootstrap node. When a node receives a STARTING_SEGMENT_REQUEST, the following cases can happen:

- The node is *Free*: it should set itself to *CandidateStarting* and forward the packet along its free links, using broadcasting and marking these links as *tvisited* with *segID* found in the packet.
- The node is *CandidateStarting* with the same *segID* and *src_id* of the packet is equal to the node id: this means that the node was the initiator of the request, thus a starting segment has just been found and should be confirmed sending back a STARTING_SEGMENT_CONFIRM packet.
- The node is *CandidateStarting*, with the same *segID* of the packet but the *src_id* field is different from the node id: since the node is candidate with the same id, it means that it already accomplished the task of propagating that kind of packets, thus can simply ignore the event dropping the packet.
- The node is *Candidate*, with a different *segID*: this simply means that the STARTING_SEGMENT_REQUEST just received is deprecated, because the node has already accepted a non-starting segment request originated from another *Assigned* node.
- The node is *Assigned*, with a different *segID*: same as the previous case.

4.2. Receiving a SEGMENT_REQUEST

When a node receives a SEGMENT_REQUEST, there are the following cases:

- The node is *Assigned*: a segment should be confirmed sending back a SEGMENT_CONFIRM packet.
- The node is *Candidate*: it should discard the packet sending back a SEGMENT_CANCEL.
- The node is *Free* and has free links: it marks itself as *Candidate* and forwards the SEGMENT_REQUEST to one of its free links, according to some internal ordering.
- The node is *CandidateStarting*: same as being *Free*, since a segment request circulating in the network indicates that the starting segment process has been completed.

Note that the main difference between confirming a STARTING_SEGMENT_REQUEST and confirming a SEGMENT_REQUEST is that in the first case the node itself is included in the segment.

4.3. Receiving a SEGMENT_CONFIRM

When the node status is *Candidate* with a *segID* corresponding to the one indicated in the packet, the node set itself to *Assigned* to the segment *segID*. Further, it should forward this packet to the link where the original SEGMENT_REQUEST packet came from, so that all candidate nodes can learn the id of the segment they belong to. Then, *LED* should be updated from *tvisited* to *visited*.

4.4. Receiving a SEGMENT_CANCEL

When a node receives a SEGMENT_CANCEL packet it means that the research for a segment along that path was unsuccessful. If the node still has some other free link to try, it should forward a SEGMENT_REQUEST to the those links. A node forwards back the SEGMENT_CANCEL packet along the link that originated the SEGMENT_REQUEST packet *only* when there are no more free links to try. If this is the case, the node modifies its status from *Candidate* to *Free* and forwards the SEGMENT_CANCEL packet to the link from which the request was received. The process stops when the SEGMENT_CANCEL packet reach the starting node that originated the request.

4.5. Intra-node vs inter-node parallelism

In the processes described above, we assumed that each node in the *Assigned* status can start a segment searching process by injecting a SEGMENT_REQUEST. However, different choices could have been made when defining DiSR, these choices are strictly related to which kind and level of parallelism should be supported. The approach adopted currently is that, although the nature of DiSR is intrinsically parallel, the use of parallelism should not make things work in a too complex/uncontrollable way. In other words, DiSR is parallel when needed, but it does not exploit parallelism as an "improving feature". Thus, when not needed, things should be serialized. For example: a node with free links could start several segment requests associated to the same *segID*, one for each free link, but serializing this operation, by investigating the free links in order, could be a simpler solution. We refer to this DiSR design choice saying that we avoid *intra-node parallelism*. Note that, conversely, avoiding *inter-node parallelism* could be more complex than allowing it: for example, we can imagine the effort generated when trying to coordinate nodes so that a unique segment searching process is actually running in the whole subnet. Thus,

Please cite this article in press as: Catania V et al. Distributed topology discovery in self-assembled nano network-on-chip. Comput Electr Eng (2014), http://dx.doi.org/10.1016/j.compeleceng.2014.09.003

in contrast to the *intra-node parallelism*, the *inter-node parallelism* is a structural property of the DiSR algorithm and should not be avoided.

5. Simulation and results

In this section we will test the proposed DiSR approach to demonstrate its effectiveness and compare it against a topology agnostic approach based on spanning trees and broadcasting [16], to measure how DiSR performs in covering the network structure. Note that a direct comparison against the SR segment-based approach is not addressed here since, how described in Section 1, our scenario assumes the non-feasibility of a centralized approach. However, all the properties of the centralized approach should be considered as preserved for all the nodes reached by the distributed segment coverage of DiSR. The main points that remain to be addressed are:

- how DiSR compares against the state-of-art tree based approach applied to an equivalent scenario,
- how DiSR scales with large networks,
- the feasibility of actual implementation of the required hardware on the limited node size assumed.

5.1. Nanoxim environment

In order to quantitatively and qualitatively evaluate the proposed approach a specific simulation environment has been developed, resulting in the open-source and freely available project called Nanoxim [23]. Nanoxim is a SystemC tool based on an almost rewritten version of the Noxim Network-on-Chip simulator [24]. While some complex features have been removed (e.g. wormhole, congestion/topology aware routing and selection strategies) new features specifically tailored for the nanoscale scenario were introduced, e.g. the ability to simulate a random network, the implementation of DiSR to obtain the segment topology and the support for defective links and nodes.

5.2. Experimental setup

The following parameters have been taken into account while performing the DiSR simulation:

- *Size of the network*: number of nodes, on a range from 10×10 to 100×100 sized networks.
- *% defective nodes*: the probability that a node is not working, thus having all its links not able to be utilized during DiSR setup.
- *Bootstrap node*: the node, from upper layer, that injects the DiSR process. When not explicitly investigating the impact of each single bootstrap choice, a set representative regions have been considered, e.g. the central part of the network and the corners.

To present the results, the following evaluation metrics have been adopted:

- *Node coverage*: this is the fraction of nodes that are assigned to a segment. In the ideal case, all the non defective nodes should be assigned, so this metric is useful to show how some disconnected regions can negatively impact on the whole DiSR effectiveness.
- *Latency*: this measures how the cycles required to complete the segment assignment scale for increasing network sizes and defect rates.
- *Bootstrap node effect*: this evaluates the impact of the chosen bootstrap node on the node coverage.

Since the distribution of defects and thus the resulting topology is randomly generated, a set of simulations with different seeds has been run for each system configuration. We found that 20 repetitions are required in order to obtain statistically significant results.

5.3. Results

In this section we analyze the results in terms of node coverage and latency with different network sizes, defect rates and bootstrap injection points. In particular Fig. 5 shows node coverage for DiSR and Reverse Path Forwarding (RPF) tree based approach (as adapted in [25]) respectively. While the first aim of DiSR is not to reach the optimal coverage, we still can observe quite good performances as compared to the tree based approach. For low defect rates, i.e. below 10%, both approaches reach near ideal coverage for each of the tested network sizes. Higher defect rates seems to show more variance for DiSR. For example, at the high 20% rate DiSR ranges from 60% to 75% while RPF remains stable at 75%. However, we can consider this coverage sensibility at higher defect rates as still acceptable for this first, not yet optimized version of the proposed approach. Note that defect rates beyond 25% lead to many disconnected regions of nodes that DiSR currently cannot handle. This threshold can be intuitively related to the fact that, in the topologies simulated, a node has (on average) a

Please cite this article in press as: Catania V et al. Distributed topology discovery in self-assembled nano network-on-chip. Comput Electr Eng (2014), http://dx.doi.org/10.1016/j.compeleceng.2014.09.003

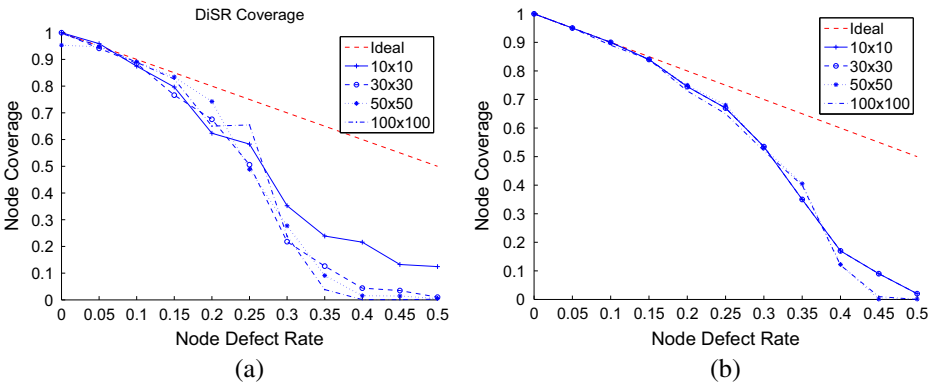

Fig. 5. DiSR (a) and RPF (b) node coverage.

cardinality of 4 connections to other nodes. This means that with a defect rate bigger than 25% a node has at least one of its four paths as not-working, and this applies (on average) for each node, leading to higher probability of consecutive disconnected nodes, which eventually create disconnected regions. For example Fig. 6 shows a 30 × 30 network with a 25% defect rate in which the bottom-left part cannot be reached due to disconnected regions. Note that the remaining defective nodes belonging to connected regions are successfully surrounded by DiSR coverage; in any case, these defect levels should be considered as worst case scenarios, so the achieved coverage of 0.5 is a satisfying result for this first version of DiSR. On the other hand, the network size seems to have a limited impact when defect rate does not introduce too many disconnected regions.

The number of cycles required to complete segment mapping process is shown in Fig. 7. In this case the comparison against the tree-based approach shows better (lower) values at different defect rates. Rather than the absolute numbers, what is more interesting to observe is how DiSR latency scales with network size. For example, moving from 900 to 2500 nodes, at an average defect rate of 0.15, leads to an increase of latency from 3000 to 4500 cycles. It should be noticed also how for DiSR latency is increasing until the threshold of 0.25 is reached, meaning that the completion of the process is more and more difficult due to the missing paths, but DiSR is still able to finish the segment discovery using the retry/cancel mechanisms described in the previous section. This initial behavior is not reported in the RPF based approach, which does not use the same retry/cancel approach as DiSR and then suffers higher latency values.

After the 0.25 threshold, the impact of disconnected regions becomes predominant and both approaches become faster in completing the covering process, since far less nodes can be actually reached.

Finally, Fig. 8 visually represents the stability of the approach against a different bootstrap node choice in a 10 × 10 network. This is an important aspect to evaluate considering that one of the main advantages of DiSR against all the tree-based approaches is the possibility of choosing whatever bootstrap node, without having to care about the role assumed in the future by the chosen root node. In other words, after that the segments have been established, the bootstrap node is like every other node, i.e. it is not center of a structure, and it is not an hotspot for the traffic distribution. The results in terms of coverage, shown for low, medium and high defect profiles, demonstrate a relatively limited impact of the bootstrap node choice in the low/medium scenarios, while a 30% instability is found for very high defect rates. This also sounds acceptable, since when a lot of defective nodes are present, the particular position of the bootstrap node could lead to a completely different evolution in the DiSR setup process.

5.4. Other optimizations

Some optimization parameters, which demonstrated to improve the DiSR results, have been fixed to some reasonable default values (see below) and are not subject to further investigations in this paper; once again the focus here is not the optimal setup of segments, but just demonstrating a working approach. These parameters are:

- *cycle_links*: max number of retries across the set of links of each node. While searching for a free link due to an incoming `SEGMENT_REQUEST`, the request itself is canceled after a given number of tries. This gives to the preceding node on the path the chance to test a different route instead of waiting indefinitely. Default value is set to 1, i.e. each link is tried one time.
- *time to live (TTL)*: when a segment request is canceled along a certain path (using a `SEGMENT_CANCEL`), a TTL field is decreased in the request packet, so that requests that have been denied too much times are canceled, even if free links are available. This is not only an optimization to shorten requests' processes, but also solves some critical situations in which request packets are blocked in routing loops in a particular path.
- *bootstrap_timeout*: number of time units that a bootstrap node should wait before assuming that a livelock in the starting segment process has occurred. In the worst case, we can imagine that the longest path required is the one returning to bootstrap node after having traveled across all the links. So, although this is just an extreme situation, a good upper limit can be safely be set to $N \times N$.

Please cite this article in press as: Catania V et al. Distributed topology discovery in self-assembled nano network-on-chip. Comput Electr Eng (2014), http://dx.doi.org/10.1016/j.compeleceng.2014.09.003

Fig. 6. Covered regions in a 30 × 30 network with 25% of defects.

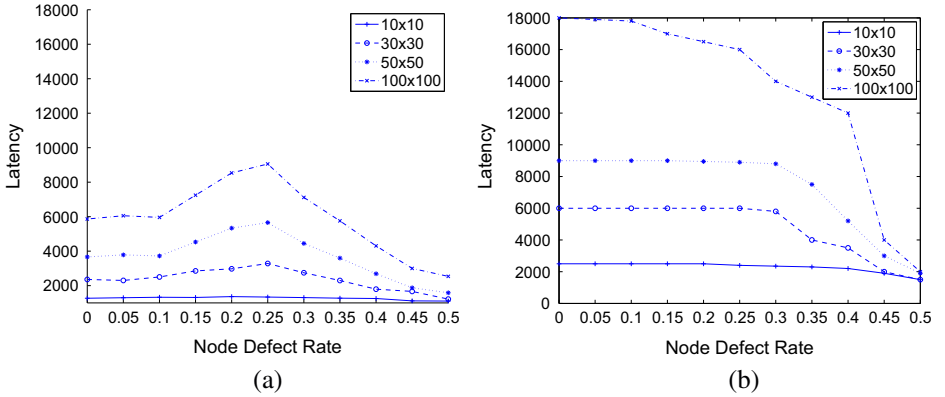

Fig. 7. Latency of DiSR (a) vs tree based RPF (b).

Please cite this article in press as: Catania V et al. Distributed topology discovery in self-assembled nano network-on-chip. Comput Electr Eng (2014), http://dx.doi.org/10.1016/j.compeleceng.2014.09.003

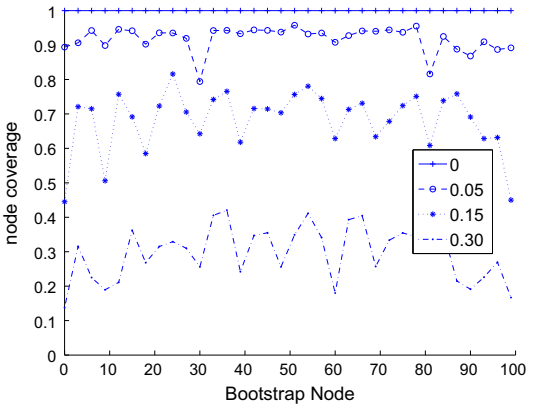

Fig. 8. Effect of bootstrap node.

- *bootstrap_immunity*: in order to avoid the failure of the whole DiSR setup process, a bootstrap node should not have defective links. Enabling this optimization, a bootstrap node is immune to defects. We may think about a pre-bootstrap phase that properly selects (from upper layer via) a bootstrap node which is tested as properly connected. We enabled this optimization, however empirical tests have shown us that only simulations using bootstrap nodes placed on edges would be heavily affected by similar issues since these nodes start with a lower number of links, e.g. corner nodes could only have two connected directions, so even a single defective link could prevent a `STARTING_SEGMENT` packet from coming back to the bootstrap node to close the loop and create the first segment.

6. Hardware implementation

A possible hardware implementation of DiSR algorithm will be described in this section. Fig. 9 depicts the general structure of a node in the particular scenario of DNA nano Network-On-Chip. In particular, such a node is composed of the following fundamental elements:

- **I/O buffers-transceivers:** These elements, one for each port, are responsible for data transmission with neighbors nodes. The packets received or sent are stored in specific buffers named input and output buffers respectively.
- **Switch matrix:** Driven by a switch controller, it enables reciprocal connections among devices inside the node. Before segments' creation, this controller receives information from DiSR block. After this phase, the switch controller will be driven by the block implementing the routing algorithm. The *Switch Matrix* is essentially composed of a series of multiplexers and demultiplexers.

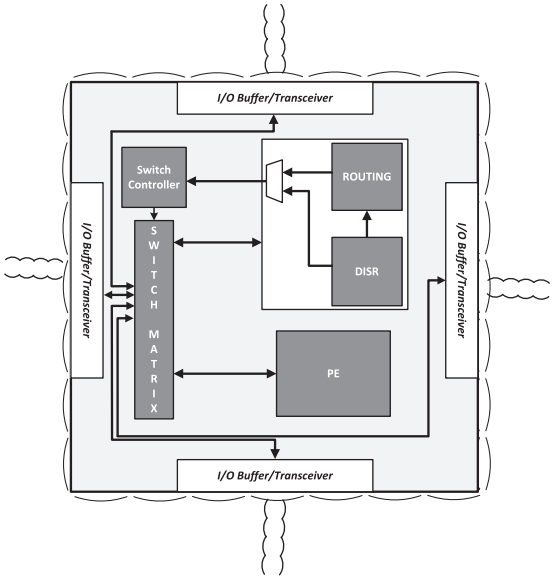

Fig. 9. A possible node structure tailored for a DNA nano network-on-chip.

Please cite this article in press as: Catania V et al. Distributed topology discovery in self-assembled nano network-on-chip. Comput Electr Eng (2014), http://dx.doi.org/10.1016/j.compeleceng.2014.09.003

- **Processing Element (PE):** It is strictly related to the functionality and role that the node cover inside a given network (e.g. being a computation or storage node).
- DiSR**-routing:** The DiSR block contains all the hardware, control logic and configuration registers, needed by the implementation of the proposed approach. The routing algorithm receives information from DiSR which indicates the status of segments related to a particular node. Routing operations will take into account these information to obtain deadlock freedom. Both Routing and DiSR are connected to the *Switch Matrix* in order to receive packets from the input buffer. Since the packets are processed one at a time, a specific arbiter should be present within the *Switch Matrix* controller.

6.1. DiSR architecture

To give a basic estimate of the overhead needed, we focus on DiSR- specific components namely the control logic and configuration registers needed to implement DiSR. Fig. 10 shows an architecture sketch, which mainly consists in the following building elements:

- **DBS block:** It takes trace of the DBS state machine, consisting of a 3-bit register (to cover six DBS values) and the required combinational logic. This block receives signals from stored packets in order to decode the packet type, and from the control circuitry to change its status. The bootstrap node is selected by setting the boot signal to high during the initialization phase.
- **LED registers:** A set of registers stores the *LED* defined in Section 3. In the link_visited[] table, we defined two specific registers named *tflag* and *busy*. While the first one indicates if the specific port of a segment is already candidate or assigned, the second one records if a specific port is already *visited* or *tvisited*.
- **Control circuitry:** This circuitry decodes incoming packet, LED registers and DBS, updating them when required. This block drives LED register according with *write enable (WE)* or *read enable (RE)* signals. Then, the resulting *Ctrl-Out* drives the other communication resources to actuate DiSR routing operations.
- **Packager:** Essentially it is a set of registers and multiplexers. Starting from an incoming packet stored in the input buffer, it updates packet's content before retransmission to a destination node. In some cases, this circuitry builds the packet from scratch e.g. when a node is bootstrap and for the first time it should inject a *STARTING_SEGMENT_REQUEST*. Fig. 11 depicts the architecture of this block. When a packet is created for the first time, the multiplexers driven by the controller, will select information incoming from an internal node such as the *src_id*. In this situation the TTL field should be reset to its initial value and the *segID* is composed of the node ID and by the output port identification number.

While the elements mentioned above are part of the DiSR circuitry, the other devices depicted in Fig. 10 like the multiplexer and demultiplexer are implemented outside DiSR. In particular, such components will be inside the *Switch Matrix* (see Fig. 9).

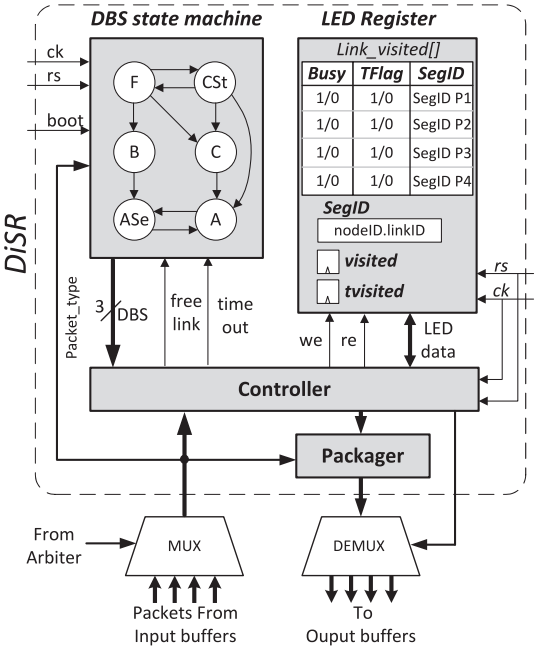

Fig. 10. DiSR block architecture.

All the blocks described above, react with the rising edge of the *clock signal*. An asynchronous system *reset* signal is also present to restore all registers to their default values. When this signal is set, DBS state machines for each network node are set as *Free*. Fig. 12 shows the pipeline structure of the DiSR architecture. Packets incoming from neighbor nodes (or generated locally) are sent to the DBS state machine in order to ensure the status commutation. While DBS updates its state, in the next clock cycle the control circuitry will update LED registers status and will drive the *Switch Matrix* in order to send the packets to the right output buffer.

6.2. Synthesis results: timing, area and power consumption considerations

As discussed above, one of the main design challenges of DNA Self-Assembled systems is the limited number of resources available to implement both computations and routing decisions for each node. Assuming a budget of 10^4 CNFETs for each network's node [11] we estimated the resources required to implement the entire DiSR block. The RTL description of the circuitry described in the last section (with an hardware description language) has been written and synthesized at gate-level using *Synopsys Design Compiler* with a generic technology library (GTECH). Considering the specific layout of each single logic element (NAND, full-adder, latch etc.), it has been possible to get a rough estimation of the number of transistors necessary to implement DiSR logic. Fig. 13 shows the results of synthesis in terms of number of devices (CNFETs) versus the number of network nodes while the network scales up from 10×10 to 100×100 nodes.

In particular, Fig. 13 shows that the proposed implementation occupies about 17.5% of the node budget. The main contribution is due to the controller circuitry followed by LED registers. Further, more than the absolute number of devices itself, it is interesting to observe that the complexity of the circuitry necessary to implement DiSR increases with a slowly growing trend. This is due to the relatively simple logic of DiSR which is almost coded with scalable storage structures. For example, the number of registers implementing the *link_visited[]* and *link_tvisited[]* table follow the logarithmic function $N_{reg} = N_{port} \cdot log_2(N)$ where *Nport* is the number of the routers ports and N is the number of networks nodes.

In order to have an idea of the maximum working clock frequency for the implemented circuitry, timing results can be obtained from a gate-level netlist. Since a complete technology library useful for commercial synthesis tool (e.g. Design Compiler) is not available for this particular technology, we have synthesized the RTL description of DiSR, with a standard 32 nm CMOS library, obtaining a delay of about $\tau_{DiSR} = 10$ FO4 (fan-out of four). Considering the results obtained in [26], which reports the ratio in terms of FO4 between a standard 32 nm CMOS technology and a carbon nanotube one, a rough delay estimation can be calculated. In particular in such work has been reported a ratio of:

$$\frac{FO4_{CMOS}}{FO4_{CNFET}} = 2 \tag{1}$$

Considering that FO4 CMOS is about 30 ps, and the results of last equation, a FO4 for the CNFET case is equal to about 15 ps, the working frequency can be computed as:

$$f_{clk} = \frac{1}{T_{max}} = \frac{1}{\tau_{DiSR} \cdot FO4_{CNFET}} = \frac{1}{10 \cdot 15 \cdot 10^{-12}} = 6.6\,GHz \tag{2}$$

where τ_{DiSR} is expressed in terms of FO4.

With regard to the power consumption of the whole set of DiSR devices: due to the fact that DiSR circuitry will be active only once during system startup, an accurate power analysis related to switching activity is not provided in this work. After the setup phase, when all segments are mapped, the DiSR block will stop its activity passing all the information related to segments to the routing algorithm. For this reason the dynamic power falls to zero and the only consumption is due to leakage effects. A comparison between the proposed circuitry and the state of the art implementation for the RPF algorithm, presented in [25], can show that there is not an appreciable discrepancy in terms of transistors cost and power consumption. In facts, RPF requires an overhead of about 1692 CNFET for a 100×100 network. Furthermore, regarding the power consumption, also the RPF approach has a setup phase beyond which, the consumption tends to zero.

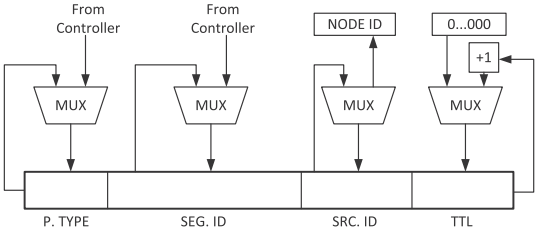

Fig. 11. Packager block.

Please cite this article in press as: Catania V et al. Distributed topology discovery in self-assembled nano network-on-chip. Comput Electr Eng (2014), http://dx.doi.org/10.1016/j.compeleceng.2014.09.003

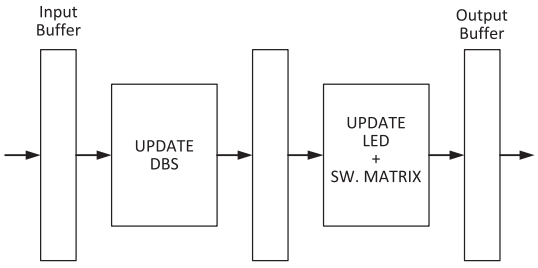

Fig. 12. Pipeline description of DiSR operations.

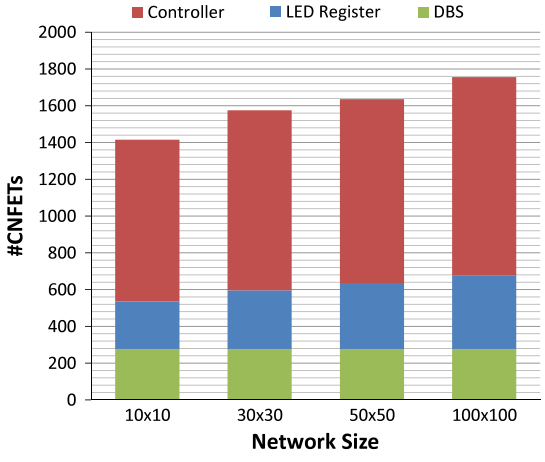

Fig. 13. Network size vs number of CNFETs necessary to implement the DiSR circuitry.

7. Conclusions

In this work, we presented DiSR, an initial attempt of achieving a segment based topology discovery in a distributed self-assembled nanoscale scenario. We demonstrated how DiSR can accomplish the aim of finding a segment coverage of the network without requiring any centralized approach that would request the graph topology as input. A draft hardware implementation has been presented to evaluate the impact on the limited node size typical of the assumed scenario. From a high-level perspective, future works will focus on investigating DiSR resulting networks in order to support the execution of massively parallel applications, while continuing to develop a detailed low level hardware implementation on a DNA grid using the appropriate nano device models.

References

[1] ITRS 2013 edition. International technology roadmap for semiconductors; 2013.
[2] Kavi K, Nwachukwu I, Fawibe A. A comparative analysis of performance improvement schemes for cache memories. Comput Electr Eng 2012;38(2):243–57. http://dx.doi.org/10.1016/j.compeleceng.2011.12.008.
[3] Sibai FN. Design and evaluation of low latency interconnection networks for real-time many-core embedded systems. Comput Electr Eng 2011;37(6):958–72. http://dx.doi.org/10.1016/j.compeleceng.2011.08.008.
[4] Yan H, Park SH, Finkelstein G, Reif JH, LaBean TH. Dna-templated self-assembly of protein arrays and highly conductive nanowires. Science 2003;301(5641):1882–4.
[5] Patwardhan JP, Dwyer C, Lebeck AR, Sorin DJ. Nana: a nano-scale active network architecture. J Emerg Technol Comput Syst 2006;2(1):1–30. http://dx.doi.org/10.1145/1126257.1126258.
[6] Pistol C, Chongchitmate W, Dwyer C, Lebeck AR. Architectural implications of nanoscale integrated sensing and computing. In: Proceedings of the 14th international conference on architectural support for programming languages and operating systems. ASPLOS, vol. XIV. New York (NY, USA): ACM; 2009. p. 13–24. http://dx.doi.org/10.1145/1508244.1508247.
[7] Haselman M, Hauck S. The future of integrated circuits: a survey of nanoelectronics. Proc IEEE 2010;98(1):11–38. http://dx.doi.org/10.1109/JPROC.2009.2032356.
[8] Bachtold A, Hadley P, Nakanishi T, Dekker C. Logic circuits with carbon nanotube transistors. Science 2001;294(5545):1317–20.
[9] Cui Y, Lieber CM. Functional nanoscale electronic devices assembled using silicon nanowire building blocks. Science 2001;291(5505):851–3.
[10] Seeman NC. Dna engineering and its application to nanotechnology. Trends Biotechnol 1999;17(11):437–43.
[11] Liu Y, Dwyer C, Lebeck AR. Routing in self-organizing nano-scale irregular networks. J Emerg Technol Comput Syst 2008;6(1):3:1–3:21. http://dx.doi.org/10.1145/1721650.1721653.
[12] Mejia A, Flich J, Duato J, Reinemo S-A, Skeie T. Segment-based routing: an efficient fault-tolerant routing algorithm for meshes and tori. In: International parallel and distributed processing symposium, Rhodos, Grece; 2006.

Please cite this article in press as: Catania V et al. Distributed topology discovery in self-assembled nano network-on-chip. Comput Electr Eng (2014), http://dx.doi.org/10.1016/j.compeleceng.2014.09.003

[13] Abbas A, Ali M, Fayyaz A, Ghosh A, Kalra A, Khan SU, et al. A survey on energy-efficient methodologies and architectures of network-on-chip. Comput Electr Eng 2014(0). http://dx.doi.org/10.1016/j.compeleceng.2014.07.012.
[14] Skeie T, Lysne O, Flich J, Lopez P, Robles A, Duato J. Lash-tor: a generic transition-oriented routing algorithm. In: Proceedings of the tenth international conference on parallel and distributed systems, 2004 (ICPADS 2004). IEEE; 2004. p. 595–604.
[15] Koibuchi M, Jouraku A, Watanabe K, Amano H. Descending layers routing: a deadlock-free deterministic routing using virtual channels in system area networks with irregular topologies. In: Proceedings of the international conference on parallel processing, 2003. IEEE; 2003. p. 527–36.
[16] Patwardhan JP, Dwyer C, Lebeck AR, Sorin DJ. Evaluating the connectivity of self-assembled networks of nano-scale processing elements. In: IEEE international workshop on design and test of defect-tolerant nanoscale architectures (NANOARCH 05); 2005.
[17] Sancho JC, Robles A, Duato J. A flexible routing scheme for networks of workstations. In: High performance computing. Springer; 2000. p. 260–7.
[18] Gomez ME, Duato J, Flich J, Lopez P, Robles A, Nordbotten NA, et al. An efficient fault-tolerant routing methodology for meshes and tori. Comput Architect Lett 2004;3(1). 3-3.
[19] Koibuchi M, Matsutani H, Amano H, Pinkston TM. A lightweight fault-tolerant mechanism for network-on-chip. In: Proceedings of the second ACM/IEEE international symposium on networks-on-chip. IEEE Computer Society; 2008. p. 13–22.
[20] Flich J, Duato J. Logic-based distributed routing for nocs. Comput Architect Lett 2008;7(1):13–6.
[21] Liu C, Zhang L, Han Y, Li X. A resilient on-chip router design through data path salvaging. In: Proceedings of the 16th Asia and South Pacific design automation conference. IEEE Press; 2011. p. 437–42.
[22] Ebrahimi M, Daneshtalab M, Plosila J, Tenhunen H. Minimal-path fault-tolerant approach using connection-retaining structure in networks-on-chip. In: 2013 Seventh IEEE/ACM international symposium on networks on chip (NoCS); 2013. p. 1–8. http://dx.doi.org/10.1109/NoCS.2013.6558401.
[23] Patti D. Nanoxim: nano network-on-chip simulator. <http://https://code.google.com/p/nanoxim/>.
[24] Fazzino F, Palesi M, Patti D. Noxim: network-on-chip simulator. <http://noxim.sourceforge.net>.
[25] Jaidev Patwardhan P, Chris D, Alvin RL. Design and evaluation of fail-stop self-assembled nanoscale processing elements. In: IEEE international workshop on design and test of defect-tolerant nanoscale architectures (NANOARCH06); 2006.
[26] Deng J, Patil N, Ryu K, Badmaev A, Zhou C, Mitra S, Wong HSP. Carbon nanotube transistor circuits: circuit-level performance benchmarking and design options for living with imperfections. In: Solid-state circuits conference, 2007 (ISSCC 2007). Digest of technical papers. IEEE International; 2007. p. 70–588.

Vincenzo Catania received the Laurea degree in Electrical Engineering from the University of Catania, Italy, in 1982. Until 1984, he was responsible for testing microprocessor system at STMicroelectronics, Catania. He is a full professor of computer science. His research interests include performance and reliability assessment in parallel and distributed system, VLSI design, low power design, and fuzzy logic.

Andrea Mineo received the BSc and MSc degrees in Electronic Engineering from the University of Catania, Italy, in 2010 and 2013, respectively, and is currently pursuing the PhD degree in Systems, Energy, Computer and Telecommunications Engineering at the University of Catania. His current research interests are VLSI systems, Network-on-Chip architectures and emerging interconnect technologies for on-chip networks.

Salvatore Monteleone is a post-doctoral research assistant at the Department of Electrical, Electronic and Computer Engineering, University of Catania, Italy. He obtained the BSc (2007) and MSc (2010) in Computer Engineering at University of Catania where he also completed the PhD course in Communications and Computer Engineering (2014). His interests comprehend cooperative systems, user-centric systems and Network-on-Chip architectures.

Davide Patti received the Laurea and PhD degrees in computer engineering from the University of Catania, Italy, in 2003 and 2007, respectively. His research focuses on Design Space exploration of VLIW systems, Network-on-Chip architectures, DNA self-assembled networks and Human-computer interfaces. He is currently a research assistant at University of Catania.

Please cite this article in press as: Catania V et al. Distributed topology discovery in self-assembled nano network-on-chip. Comput Electr Eng (2014), http://dx.doi.org/10.1016/j.compeleceng.2014.09.003