Computers and Electrical Engineering 000 (2016) 1–12

Contents lists available at ScienceDirect

Computers and Electrical Engineering

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

Reconfigurable fault tolerant routing for networks-on-chip with logical hierarchy☆

Gert Schley[a], Ibrahim Ahmed[a,b], Muhammad Afzal[a,c], Martin Radetzki[a,*]

[a] Embedded Systems Group, University of Stuttgart, Pfaffenwaldring 5b, Stuttgart 70569, Germany
[b] Dept. of Electrical and Computer Engineering, University of Toronto, 10 King's College Road, Toronto, ON M5S 3G4, Canada
[c] Altium Europe GmbH, Philipp-Reis-Strasse 3, Karlsruhe 76137, Germany

ARTICLE INFO

Article history:
Received 29 April 2015
Revised 16 February 2016
Accepted 16 February 2016
Available online xxx

Keywords:
Networks-on-chip
Hierarchy
Routing
Fault tolerance
Reconfiguration

ABSTRACT

This paper presents a reconfigurable fault tolerant routing for Networks-on-Chip organized into hierarchical units. In case of link faults or failure of switches, the proposed approach enables the online adaptation of routing locally within each unit while deadlock freedom is globally ensured in the network. Experimental results of our approach for a 16 × 16 network show a speedup by a factor of almost four for routing reconfiguration compared to the state-of-the-art approach. Evaluation with transient faults shows that a dedicated reconfiguration unit enables successful reconfiguration of routing tables even in case of high error probabilities.

1. Introduction

The ongoing technology scaling allows an increasing number of cores to be implemented on a single chip, e.g. Intel's Xeon Phi Coprocessor [1] or Tilera's Tile-MX multicore processor [2]. As this scaling trend continues [3], future multiprocessor systems will feature hundreds of cores on a single chip.

The increasing size of on-chip systems poses a new challenge to Networks-on-Chip (NoCs). Mechanisms implemented in an NoC, such as table-based routing, work well for small systems but do not scale for bigger systems. A possibility to cope with scalability problems is the introduction of a hierarchical structure to NoCs. A hierarchical NoC can be obtained by constructing its topology using subnetworks or by segmenting a given topology into logical units. Both subnetworks and logical units enable formerly global mechanisms to be applied locally thus reducing their complexity. Compared to subnetworks, logical units have the advantage that they can be applied without changing an existing topology. Typically, network nodes (switch + core) are grouped into a logical unit if they are part of the same task and share a spatial relation.

The downside of technology scaling is the increased probability of occurrence of permanent faults in an NoC due to manufacturing inaccuracies [4] or wear-out effects such as electromigration [5] that emerge during system operation. The failure of links or switches due to permanent faults results in an altered network topology. In such a case, static routing can no longer maintain connectivity between system components. For this reason, it is crucial that the routing is adapted to the new network situation to enable packets to circumvent faulty components.

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

* Corresponding author. Tel.: +4971168588270.
E-mail addresses: gert.schley@informatik.uni-stuttgart.de (G. Schley), ibrahimai.ahmed@mail.utoronto.ca (I. Ahmed), muhammad.afzal@outlook.de (M. Afzal), martin.radetzki@informatik.uni-stuttgart.de (M. Radetzki).

http://dx.doi.org/10.1016/j.compeleceng.2016.02.013
0045-7906/

In this paper we present a reconfigurable fault tolerant routing approach based on *Up/Down* routing [6] for large scale NoCs with logical hierarchy. It enables the routing to be adapted locally within each hierarchical unit in case of permanent faults while deadlock freedom is guaranteed globally. Our approach can be applied to any number of hierarchy levels.

The remainder of the paper is organized as follows: In Section 2 related work is discussed. Section 3 contains a formal introduction to NoC topologies as well as an introduction to *Up/Down* routing. In Section 4 we present our hierarchical network concept. Our hierarchical routing is presented in Section 5 and the reconfiguration process in Section 6. Evaluation results are discussed in Section 7. Section 8 concludes the paper.

2. Related work

In this section, we focus on work related to routing reconfiguration. Related work dealing with hierarchical topologies and hierarchical routing is presented in [7].

A reconfigurable scheme for source based routing is presented in [8]. If a source cannot reach a destination, it floods a *path request* through the network. Each node stores the port via which the request was received in a table. When the request reaches the destination, a packet is sent back to the source using the reverse path recording that path by means of the table entries. At the source the recorded path is checked in software if it contains violations of routing restrictions and if so *virtual channels* (VCs) are assigned to prevent deadlocks. Results in [8] show that about 2300 cycles are required to adapt a path of length 10 hops. However, permanent faults usually have an impact on multiple source-destination pairs and thus the path request has to be initiated for each of them. This results in a high time to adapt the routing completely. Furthermore, source based routing induces great costs for implementation of routing tables.

In [9], a centralized routing management system for high-performance networks is proposed. After the occurrence of a fault, the central manager first discovers the remaining topology and creates a topology graph with *Up/Down* directions. This graph then is used to calculate the routing for every network node. However, determining the routing for all nodes by a single manager results in a high calculation time.

A distributed routing recalculation approach for *Up/Down* routing is presented in [10]. For routing adaptation, normal operation in network is stopped and flags are broadcast by each node. *Up/Down* routing directions are assigned to each node port during the initial broadcast. Starting with the node that has detected a permanent fault, each of the n network nodes consecutively broadcasts a flag to all other nodes via dedicated signals. The reconfiguration period is divided into n portions, each with a duration of n cycles. Upon receipt of a flag, a node records the port over which the flag was received and updates its routing table accordingly. In both approaches, [10] and [9], the routing of the entire network has to be adapted. Neither [10,9], nor [8] consider hierarchical network topologies.

A fast online routing reconfiguration algorithm based on *segment-based routing* [11] to compute emergency routes in case of a fault is presented in [12]. The approach allows the adaptation of routing in segments of the network while other segments are not affected. To compute emergency routes the algorithm makes use of routing meta data calculated offline during initial routing computation. Per segment one fault can be tolerated, however, in case of a second fault, the routing and meta data for the whole network has to be recalculated offline in software.

Furthermore, approaches exist that do not require a reconfiguration phase but achieve routing adaptation by means of additional hardware logic, e.g. [13] and [14]. While [13] can tolerate the failure of three switches, [14] provides connectivity for any irregular topology derived from a non-hierarchical 2D mesh. A disadvantage of these approaches is that the hardware logic is prone to faults as well. A permanent fault in this logic leads to a routing failure. In our approach, routing is adapted in software. In general, software-based calculating provides the possibility that in case of a network node failure the calculation task can be migrated to another network node. As in case of [14], our routing provides connectivity as long as the topology is not disconnected by faults.

3. Preliminaries

3.1. NoC topology

The topology of an NoC can be represented by a directed graph $T = (N, C)$ where N is the set of network nodes and C is the set of unidirectional channels. In a typical fault free NoC topology, two connected nodes n_i, $n_j \in N$ have one bidirectional connection, i.e. $c_{i,j}$, $c_{j,i} \in C$. We refer to bidirectional connections as *links* and unidirectional connections are called *channels*. To distinguish between different nodes, each node has a unique ID.

3.2. Up/Down routing

In literature, various approaches (e.g. [10,15,16]) based on *Up/Down* routing can be found. *Up/Down* routing, as originally described in [6], ensures an inherently deadlock free routing for arbitrary network topologies.

For *Up/Down* routing calculation for a given topology T, the spanning tree S of T is determined starting from a root node. Based on spanning tree S, *Up* or *Down* direction is assigned to each channel of C. *Up* direction is assigned to channel $c_{i,j}$ if n_j has a shorter distance to the root node than n_i. If two nodes have the same distance to the root node, then *Up* direction is assigned to the channel leading to the node with the smaller ID. In all other cases, *Down* direction is assigned. To ensure

Please cite this article as: G. Schley et al., Reconfigurable fault tolerant routing for networks-on-chip with logical hierarchy, Computers and Electrical Engineering (2016), http://dx.doi.org/10.1016/j.compeleceng.2016.02.013

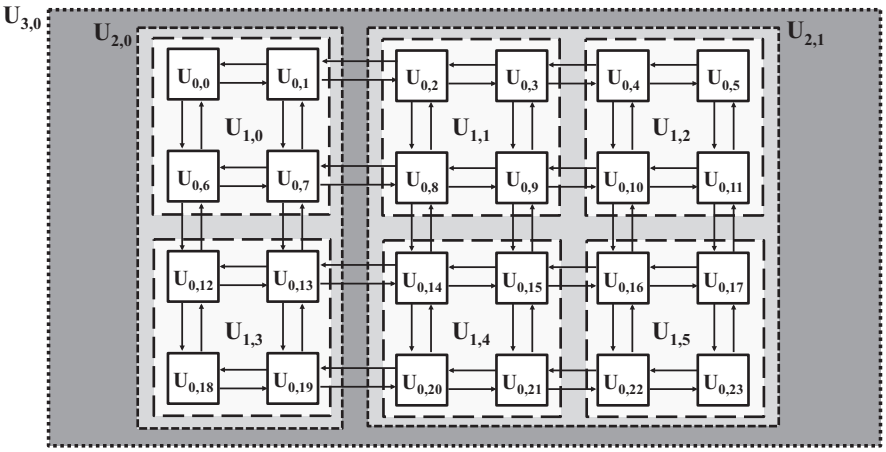

Fig. 1. Hierarchical mesh topology with $h_{max} = 3$.

deadlock freedom of *Up/Down* routing, a valid path p must consist of zero or more channels in *Up* direction followed by zero or more channels in *Down* direction [6].

4. Hierarchical network concept

Our approach is based on introducing hierarchical levels to a network topology and organizing network nodes into hierarchical units on each level. For a topology T, h_{max} hierarchy levels may be defined, where $h_{max} = 1$ corresponds to the flat hierarchy network. Each level $0 \leq h \leq h_{max}$, consists of hierarchical units, which are composed of a set of connected network nodes. Each node must belong to exactly one unit on each level. On the lowest level $h = 0$ a unit corresponds to a node.

The hierarchical units of level h are identified with $U_{h,\,i}$, where i denotes a unique ID within the level. For $h > 0$, a hierarchical unit $U_{h,i}$ aggregates one or more hierarchical units of the next lower level $h - 1$. We refer to a unit $U_{h-1,j}$ as the subunit of $U_{h,i}$ and to $U_{h,i}$ as the superunit of $U_{h-1,j}$, and we write $U_{h-1,j} \in U_{h,i}$. A hierarchical unit $U_{h-1,j}$ must be a subunit of exactly one superunit. The topology of a unit corresponds to a subgraph of T. We call channels that are connecting two hierarchical units *interconnection channels*. On highest level h_{max}, a single hierarchical unit $U_{h_{max},0}$ exists aggregating all units of the lower levels. The topology of this unit corresponds to T. An example for a hierarchically organized NoC with $h_{max} = 3$ levels is shown in Fig. 1.

To address a network node, a hierarchical address scheme is used. An address $(ID_{h_{max}-1}, ..., ID_0)$ is composed of the IDs of the hierarchical units of all hierarchy levels $h < h_{max}$, to which a node belongs to. In the case of e.g. $U_{0,21}$, the corresponding address is (1, 4, 21).

5. Hierarchical routing

The aim of our hierarchical routing approach is the online adaptation of routing in case of permanent faults in the communication structure. For our hierarchical routing, we consider permanent channel and link faults caused during manufacturing (e.g. bridging) or broken wires caused by aging. Further, we consider the complete failure of a switch. To minimize the time required for adaptation as well as the required communication overhead, the adaptation process is not performed globally in the entire network but only within the hierarchical unit that is affected by the fault, i.e. on the lowest possible hierarchy level. For example, if in Fig. 1 a channel or link connecting two subunits of $U_{1,1}$ is faulty, the routing is only adapted within $U_{1,1}$. If $U_{1,1}$ and $U_{1,2}$ cannot reach each other because both interconnection links are defective, the routing has to be adapted within unit $U_{2,1}$.

For this purpose, each unit $U_{h,i}$ implements its own internal routing used for intra-communication between its subunits. From the point of view of a subunit $U_{h-1,j}$, this is a routing on the next higher level used for inter-communication with another subunit. For communication between two nodes situated in different superunits, always the routing on highest possible level is used. Once the destination is reached on this level, routing is continued on the next lower level until, again, the destination on the lower level is reached. This is repeated until the destination node is reached.

Permanent faults cause regular topologies to become irregular, and thus, a formerly valid routing does no longer guarantee connectivity between network nodes. To guarantee a valid and deadlock-free routing for arbitrary topologies, our approach applies *Up/Down* routing on all hierarchy levels. Per level $h > 0$ two VCs are used to provide deadlock freedom. The proof for deadlock freedom of our routing can be found in [7]. The routing makes use of reconfigurable routing tables.

Please cite this article as: G. Schley et al., Reconfigurable fault tolerant routing for networks-on-chip with logical hierarchy, Computers and Electrical Engineering (2016), http://dx.doi.org/10.1016/j.compeleceng.2016.02.013

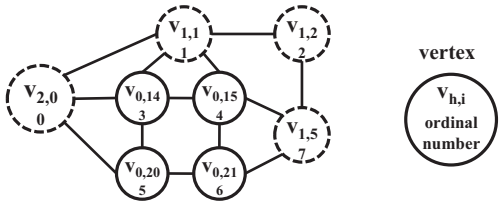

Fig. 2. Enhanced Topology Graph $ETG_{1,4}$.

5.1. Routing calculation

For online routing calculation, each hierarchical unit $U_{h,i}$ ($h \geq 1$) has one manager. A *manager* is a network node that is responsible for creating the so-called Enhanced Topology Graph (*ETG*) and to adapt the *ETG* according to fault information gathered within its unit. The creation and adaptation of *ETG*s are both software tasks performed by the manager's processing element. To handle the failure of a manager, further redundant managers may be defined that take over the task, e.g. by means of the placement method proposed by [17].

5.1.1. Enhanced topology graph

An *ETG* is an abstract topology representation used for routing calculation within a hierarchical unit. A vertex v of an *ETG* represents a hierarchical unit $U_{h,i}$. Two vertices are connected by an edge if the corresponding units are connected by at least one interconnection link. Each vertex has a unique ordinal number used by routing calculation to determine whether the corresponding unit can be reached via *Up* or *Down* direction. In accordance with *Up/Down* routing rules [6] (cf. Section 3.2), if a vertex v_k has the smaller ordinal number than v_l it is reached in *Up* direction from v_l.

The creation of *ETG*s follows a top-down manner. Starting on level h_{max}, the *ETG* of a hierarchical unit is provided to the managers of all subunits. Each manager extends the *ETG* by the abstract topology of its own hierarchical unit. This is repeated until the *ETG*s of level 1 units are all created. Details about the creation of *ETG*s can be found in [7]. The exemplary $ETG_{1,4}$ used to calculate the routing for all nodes in unit $U_{1,4}$ is shown in Fig. 2. The calculation of routing by means of ETGs is described in the following.

5.1.2. Routing table computation

$ETG_{1,i}$ on level 1 contains all necessary information to compute the table entries for all nodes within unit $U_{1,i}$. To reduce the time complexity of routing calculation, routing tables are computed in parallel by distributing the workload among the processing elements of all nodes within a unit. For this purpose, the manager sends the *ETG* to all nodes within its hierarchical unit. Hereby it is ensured that all nodes decide routings on the same basis. After having received the *ETG*, each node computes its own table entries. This results in a time complexity of routing calculation of $\mathcal{O}(|V|^2)$.

To compute the table entries for a node, paths from the corresponding vertex to all other vertices in $ETG_{1,i}$ are determined by a modified Dijkstra algorithm that in contrast to the original Dijkstra algorithm takes the *Up/Down* rule for valid paths into account. The pseudo code of the modified Dijkstra algorithm is shown in Algorithm 1. Input parameters of the algorithm are the *ETG* and the start vertex (*start_vertex*). A valid path consists of a sequence of vertices so that on each level at maximum one *Up* to *Down* turn exists. Furthermore, two consecutive vertices n_i and n_{i+1} of a valid path have either to represent units of the same level or n_{i+1} is a unit of a higher level than n_i.

To decide whether a vertex is reached via *Up* or *Down* direction, the modified Dijkstra algorithm makes use of the ordinal numbers $o(v)$ found in the *ETG* for each vertex v. After the initialization (lines 2–6), the algorithm determines if the current vertex u was reached via *Down* direction ($down_{prev}$) from its predecessor $prev[u]$ (line 9). Subsequently, the same is done for every vertex v connected to u (line 11). A vertex v can be reached from u if (line 12) either

- v and u are on the same level and this does not result in a *Down* to *Up* turn or
- v represents a unit on a higher level (*lvl*) than u.

If v represents a unit of lower level than u, it cannot be reached as this does not result in a valid path. In case v can be reached, the costs (*new_cost*) to reach v are calculated. For this purpose, the costs to reach u ($cost[u]$) are increased by one plus the weighted level of v (line 13). The higher the level of a vertex, the more network nodes it represents. The path length with the number of vertices of higher level. The weighting is done to constrain a path to consist of vertices of the lowest possible levels. In most cases, this results in shorter paths.

If the new calculated costs of v are less than the current ones, the entry of v in the queue (*sorted_Q*) is updated if available or a new one is added. Vertex u becomes a predecessor of v (lines 14–16). The steps described above are repeated until all vertices have been visited. Once all vertices have been visited, the table entries can be derived from the information stored in *prev*.

Please cite this article as: G. Schley et al., Reconfigurable fault tolerant routing for networks-on-chip with logical hierarchy, Computers and Electrical Engineering (2016), http://dx.doi.org/10.1016/j.compeleceng.2016.02.013

Algorithm 1 Dijkstra Algorithm for Hierarchical *Up*/*Down* Routing.

```
procedure DIJKSTRA_UD( ETG, start_vertex )
  for all vertices n in ETG do
    cost[n] ← ∞;   prev[n] ← none;   lvl[n] ← level of n
  end for
  cost[start_vertex] ← 0
  sorted_Q ← start_vertex
  while not all vertices visited do
    u ← sorted_Q.pop_front()
    down_prev ← (o(prev[u]) < o(u))
    for all neighbors v of u do
      down_next ← (o(u) < o(v))
      if lvl[u] = lvl[v] ∧ (¬(down_prev ∧ ¬down_next)) ∨ (lvl[u] < lvl[v]) then
        new_cost ← cost[u] + 1 + weight ∗ lvl[v]
        if new_cost < cost[v] then
          cost[v] ← new_cost;   prev[v] ← u
          add v to sorted_Q or update cost of existing entry
        end if
      end if
    end for
  end while
  return prev
end procedure
```

6. Routing reconfiguration process

This section describes the routing reconfiguration process. Please note that the test process to detect faults is out of scope of this work. We assume that in the network a test mechanism such as [18,19], or [20] exists to detect faults. We further assume that each switch communicates the availability of a channel by means of availability signals to its neighbors. If a channel is available its availability signal is set to one otherwise to zero.

The reconfiguration process of the routing within a hierarchical unit consists of three different steps. In a first step, when a permanent fault occurs, it is necessary that the manager of the unit is informed about the fault to adapt the *ETG*. In a second step, the routing for the network nodes within the hierarchical unit is calculated. Finally, the routing table of each node is updated. The different steps of the reconfiguration process are carried out on different network layers. Updating the *ETG* and routing recalculation are done in software. The communication of fault information and updating routing tables are performed by dedicated hardware units, i.e. the *Fault Information Unit* and *Reconfiguration Unit*. The switch architecture showing the interaction of these two additionally required hardware units is shown in Fig. 3.

Each switch has a status. By default, a switch is in *operative mode* in which it can receive and forward data. When a permanent fault is detected in a switch or when information about a fault in another switch of the same hierarchical unit was received, a switch changes to *maintenance mode*. A switch in maintenance mode waits for the routing to be recalculated and the routing tables to be updated. In this mode, a switch does not accept any data flits. When the routing table is updated, the switch changes back to operative mode and resumes normal operation.

6.1. Communicating fault information

The *Fault Information Unit* (FIU) is responsible to inform all network nodes within a hierarchical unit about permanent faults. For this purpose, each switch implements one FIU (cf. Fig. 3). To create information about faults, the FIU makes use of the results of a fault detection mechanism and the availability signals of every input port. In our approach, the fault information is communicated using flits. Fault information flits are created by the FIU either in case of a permanent fault or if the availability signal of a channel changes to zero. The fault information fits exactly into one flit and contains the node's ID and a fault identifier. Fault information flits are sent via an additional prioritized control VC to prevent them from being blocked by data flits in the network.

Data may be corrupted by either transient or permanent faults. To observe the presence of a permanent fault, the FIU implements a 2bit counter for each input port to count the number of reported faults. A counter is increased each time a fault has been reported by the detection mechanism. It is decreased by one if for 100 cycles no further fault has been reported for the corresponding input. If a counter reaches its maximum value, the FIU considers the corresponding input to be faulty and hence it resets the availability signal of the input channel. This indicates to the neighbor node that it must no longer use that channel. In turn, the neighbor sets the availability of the reverse channel to zero as well. In this way, the complete link between the two nodes is shut down. Note that this is required because *Up*/*Down* routing can only distinguish

Please cite this article as: G. Schley et al., Reconfigurable fault tolerant routing for networks-on-chip with logical hierarchy, Computers and Electrical Engineering (2016), http://dx.doi.org/10.1016/j.compeleceng.2016.02.013

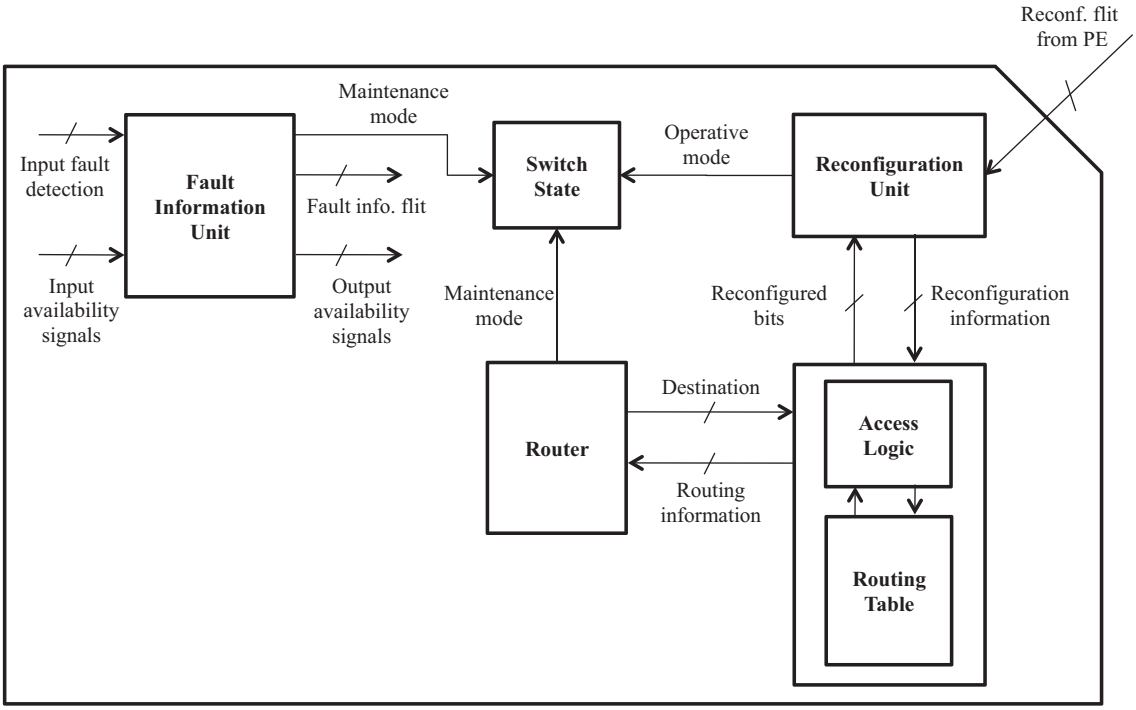

Fig. 3. Overview switch architecture for reconfiguration.

between working or not working links. If a complete node fails, this corresponds to all availability signals of the channels leading to this node are set to zero.

A fault information flit is always created by both nodes incident to a shut down link to report the fault to the manager. In this way, if both nodes are situated in different units, the manager of each unit receives the information. As a result of the link shutdown, the actual routing within the hierarchical unit can no longer be considered valid. For this reason, fault information flits are forwarded to the manager using a flooding mechanism. This ensures that every node receives the flit. However, flooding introduces a high traffic overhead to the network. To prevent an unnecessary overhead caused by fault information flits, they are only flooded within the hierarchical unit in which the fault has occurred. If a fault information flit is received by a node of an adjacent hierarchical unit it discards that flit.

If a node receives a fault information flit, its switch stops forwarding data flits and changes to maintenance mode in which it only accepts flits via the control VC. In maintenance mode, a node discards all data flits and thus the hierarchical unit is cleared of data flits eventually. This is required to ensure that no flits exist in the hierarchical unit that still use the old routing once the routing tables are reconfigured. Discarded data flits have to be retransmitted by end-to-end flow control protocol. As fault information flits are ignored by nodes in adjacent hierarchical units, they continue normal operation.

6.2. Routing recalculation

When a manager of a unit receives a fault information flit containing new fault information, the routing within its unit has to be recalculated. First, the manager extracts the fault information from the received control flit and updates the *ETG* by removing the corresponding edge in the case of a shut down link or the corresponding vertex in the case of a node failure. The manager assigns new ordinal numbers to the corresponding vertices in the *ETG*. The updated *ETG* is sent to the nodes, which calculate the new routing (cf. Section 5.1.2) and update their routing tables accordingly.

If a node can no longer reach another hierarchical unit on level h because no valid path can be found, this implies that the routing on level $h+1$ has to be adapted and thus the corresponding manager on that level has to be informed about the fault.

For example in Fig. 1, this is the case if both links connecting the two units $U_{1,4}$ and $U_{1,1}$ are shut down. The only possible communication path from $U_{1,4}$ to $U_{1,1}$ leads over $U_{1,5}$ and $U_{1,2}$. However, according to $ETG_{1,4}$ shown in Fig. 2 this would require a *Down* to *Up* turn on level 1 which is not allowed. In such a case, a node informs its manager on level 1 which in turn creates a flit with a corresponding fault information. This fault information flit is sent to the manager of the superunit. This is repeated until the manager on level $h+1$ receives the fault information flit (in the above example the manager of unit $U_{2,1}$). It then adapts its *ETG* and recalculates routing on level $h+1$. The updated *ETG* is then passed to the managers of lower levels.

Please cite this article as: G. Schley et al., Reconfigurable fault tolerant routing for networks-on-chip with logical hierarchy, Computers and Electrical Engineering (2016), http://dx.doi.org/10.1016/j.compeleceng.2016.02.013

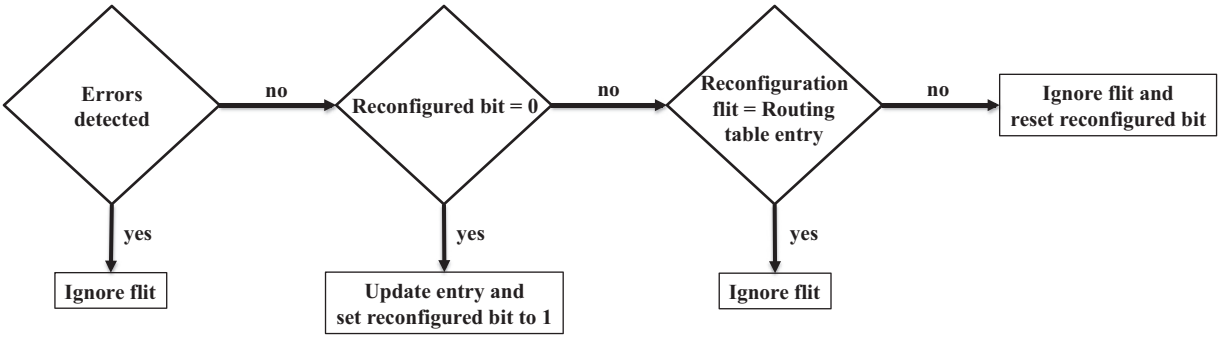

Fig. 4. Steps taken by the REU for incoming flits.

6.3. Routing table reconfiguration

After calculating its new routing table entries, a processing element sends them to its *Reconfiguration Unit* (REU) using *reconfiguration flits*. The task of the REU is to reconfigure the routing table. In our approach, each reconfiguration flit carries information that updates a single entry in the routing table. Since the switch is in maintenance mode during reconfiguration, reconfiguration flits are communicated using the control VC. When a processing element has sent all flits, it sends an *end reconfiguration flit*.

On receipt of a reconfiguration flit, the REU updates the corresponding table entry. To indicate that an entry has been updated, an extra bit (*reconfigured bit*) is added for each entry of the routing table. On the basis of these reconfigured bits, the REU determines whether the reconfiguration is complete and generates retransmission requests. If all reconfigured bits are set, the routing table has been successfully updated and the REU changes the switch state back to operative mode.

The process of updating the routing table is susceptible to e.g. transient faults as their occurrence during reconfiguration can compromise the integrity of reconfiguration flits leading to their corruption, loss, or duplication. This may cause the switch being stuck in maintenance mode due to an uncomplete reconfiguration or may lead to invalid routing table entries. A switch being stuck in maintenance mode discards all incoming data flits. Thus, all flits being forwarded to this switch will be lost. A routing table with invalid information can result in longer routes increasing the end-to-end latency, livelocks or even in deadlocks. To ensure proper reconfiguration in the presence of faults, the REU implements two modes which we denote as *local reconfiguration* mode and *remote reconfiguration* mode. Both modes are explained in the following.

6.3.1. Local reconfiguration

By default, the REU is in *local reconfiguration* mode in which the source of reconfiguration flits is the local processing element connected to the switch. As discussed in Section 6.3, faults may corrupt or duplicate reconfiguration flits, and thus, the REU must check them. The steps taken by the REU for every incoming reconfiguration flit are shown in Fig. 4. At first, the REU checks each flit for faults. For this purpose, we assume that each reconfiguration flit is augmented with a parity bit. If the flit is faulty, it is discarded. In the second step, the REU checks if the corresponding entry is already reconfigured by a flit received earlier. If it is not reconfigured, the entry is updated by the information found in the flit. Otherwise, in a third step, the REU compares the new routing information to the one found in the routing table. If both are identical, the flit is ignored and the information in the table is kept. In case both are different, the REU cannot decide which information is correct, and thus it invalidates the entry by resetting its reconfigured bit.

We now present the fault tolerance mechanism of the REU. Fig. 5 shows the state diagram of the REU in local reconfiguration mode. When the switch is in operative mode, the REU is in the *Idle* state. In this state, the REU does not expect any flits, and thus, any incoming reconfiguration flit is ignored. Whenever the switch state changes to maintenance mode, the REU invalidates all table entries and waits for the first reconfiguration flit to arrive (*Wait for start*). After receiving the first flit, the REU expects all further reconfiguration flits to arrive consecutively without much delay (*Accepting*). If either no flit is received for a certain time period (timeout) or if the REU receives the end reconfiguration flit before all entries have been updated (*Fail a*), REU sends a retransmission request to the local processing element. The REU declares the failure of local reconfiguration if during retransmission a second time-out occurs or in case of three unsuccessful reconfigurations attempts (*Fail b*). If the reconfiguration is successful, the REU sets the switch back to operative mode and returns to the *Idle* state.

In case of a permanent fault in the processing element, the network interface, or the link between the network interface and the switch, the reconfiguration process will not succeed and the REU declares the failure of local reconfiguration. However, the switch can still be functional and capable of routing flits. For this reason, the REU tries to reconfigure the routing table by means of neighbor nodes to prevent the unnecessary shutdown of a working switch. Thus, it changes to remote reconfiguration mode.

Please cite this article as: G. Schley et al., Reconfigurable fault tolerant routing for networks-on-chip with logical hierarchy, Computers and Electrical Engineering (2016), http://dx.doi.org/10.1016/j.compeleceng.2016.02.013

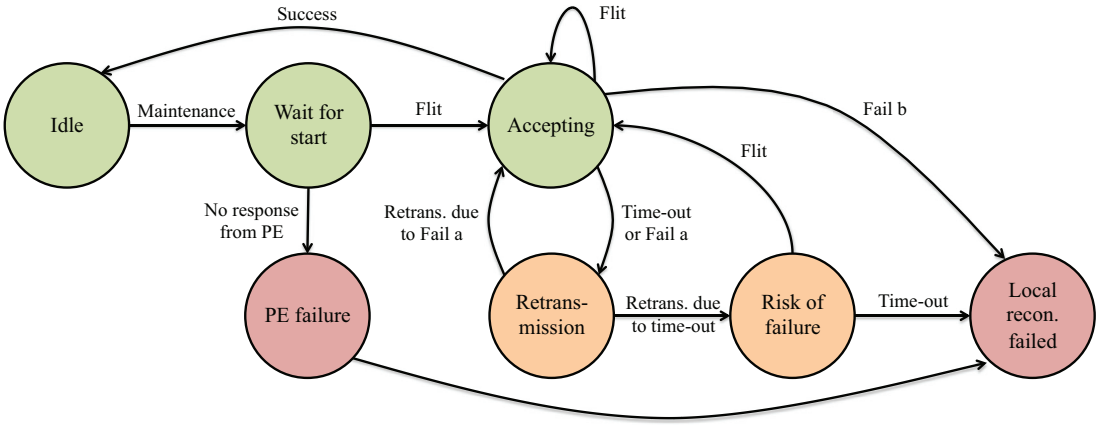

Fig. 5. State diagram of local reconfiguration.

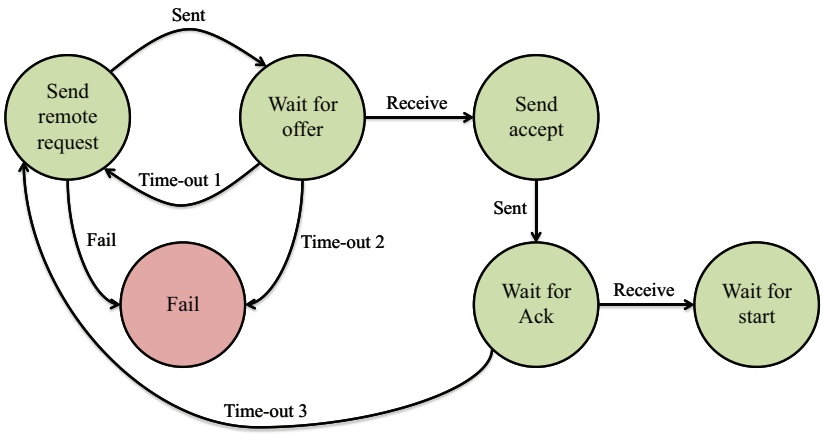

Fig. 6. State diagram of remote reconfiguration protocol.

6.3.2. Remote reconfiguration

The REU of a switch changes to remote reconfiguration mode whenever the local reconfiguration has failed. In this mode, the REU requests routing reconfiguration from one of the direct neighbors of the switch. For this purpose, the REU sends a *remote reconfiguration request* flit to all neighbors and waits for an offer (*Wait for offer*). The corresponding remote reconfiguration protocol is shown in Fig. 6. Neighbors that are allowed to serve the request send an *offer* flit. A neighbor node may serve the request if:

- The node is in operative mode, as this implies that it was able to reconfigure its own routing table.
- The node is in the same superunit on level 1. This ensures that both nodes share the same *ETG* and thus the helping node can calculate the routing table entries for the requesting node.
- The node is not busy helping any other node or has sent an offer.
- The node has not already tried to help the requesting node in the current reconfiguration process. This gives other neighbors the possibility to help.

If the REU does not receive any offer, it sends a second request. If it still does not receive any offer, the REU assumes that no neighbor can help and declares the failure of the overall reconfiguration process (*Fail*). When the REU receives offers, it accepts one of them. The chosen neighbor acknowledges the accept and starts calculating the new routing table entries. All other neighbors are released from their offer and may help another requester. When the calculation is finished, the helping node sends the reconfiguration flits to the requesting node. After the routing table has been updated, the REU informs the helping node to release it.

If remote reconfiguration is not successful, the REU requests help from one of the remaining neighbors. When the reconfiguration process is not successfully completed after requesting help from all available neighbors, the REU declares the failure of the overall reconfiguration and does not send further requests. In that case the REU informs the fault information unit which in turn generates a fault information flit to report the shutdown of the switch to the manager.

Please cite this article as: G. Schley et al., Reconfigurable fault tolerant routing for networks-on-chip with logical hierarchy, Computers and Electrical Engineering (2016), http://dx.doi.org/10.1016/j.compeleceng.2016.02.013

Table 1
Synthesis results.

Unit	Area [mm^2]
Baseline switch	0.0420
Routing table	0.0080/0.0410
Fault information unit	0.0014
Reconfiguration unit	0.0015/0.0021

Table 2
Number of routing table entries and throughput.

(s_1, s_2)	Flat	(8 × 8, 2 × 1)	(8 × 4, 2 × 2)	(8 × 4, 1 × 2)	(4 × 4, 4 × 2)	(4 × 4, 2 × 2)	(4 × 4, 2 × 1)	(4 × 2, 4 × 4)
Entries	256	66	36	36	24	22	24	24
TP	0.027	0.032	0.041	0.029	0.040	0.032	0.034	0.069
(s_1, s_2)	(4 × 2, 2 × 4)	(4 × 2, 2 × 2)	(4 × 2, 1 × 2)	(2 × 2, 8 × 4)	(2 × 2, 4 × 4)	(2 × 2, 4 × 2)	(2 × 2, 2 × 2)	(2 × 2, 2 × 1)
Entries	18	18	24	36	22	18	22	36
TP	0.029	0.029	0.026	0.057	0.029	0.035	0.033	0.042

7. Evaluation

We evaluate our reconfigurable routing regarding the required hardware implementation overhead (Section 7.1), the impact of different unit sizes on the network performance (Section 7.2), and the reconfiguration process in presence of faults (Section 7.3).

To evaluate our hierarchical routing, we have taken mesh networks with flat hierarchy as well as $h_{max} = 3$ hierarchy levels into account. For level $h = 1$ we have considered unit sizes s from 2 × 2 to 16 × 8. The size of a level 2 unit refers to the number of encapsulated level 1 units in x-dimension and y-dimension. For example, a level 2 unit size of 4 × 2 encapsulates in total eight units of level $h = 1$. The unit size s_h of level $h = 1$ and level $h = 2$ form a tuple (s_1, s_2) to which we refer to as hierarchical network configuration.

7.1. Implementation overhead

We have synthesized both the *Fault Information Unit* (FIU) and the *Reconfiguration Unit* (REU) to investigate the area overhead compared to a baseline 5-port switch with routing table. The baseline switch features five VCs and for each of them a VC buffer with three buffer slots is available per input port. The flit width used is 36 bits. For synthesis, we have used Synopsys Design Compiler with the 45 nm Nangate library [21]. To obtain the required area of routing tables for a technology size of 45 nm, we have used CACTI 5.3 from HP Labs [22]. The area results are shown in Table 1. Depending on the number of entries the routing table size varies between 0.008 mm^2 (36 entries) and 0.041 mm^2 (256 entries). Note that for less than 36 entries CACTI was not able to calculate the area as these sizes come below the minimum supported memory size of CACTI. The number of table entries for different hierarchical configurations for a 16 × 16 network is shown in Table 2.

The FIU does not depend on the number of table entries and has a fixed area requirement of 0.0014 mm^2. However, the REU depends on the number of table entries. The reason for this is the increasing complexity of logic used to analyze the reconfigured bits of the routing table and to generate retransmission requests. For 36 table entries the area requirement of the REU is about 0.0015 mm^2. In case of the baseline switch with 36 table entries, the total area overhead of FIU and REU is less than 6%.

7.2. Routing performance

To determine the impact of each unit size on the achievable throughput, we have implemented a cycle accurate NoC model. The model implements a 16 × 16 mesh topology. Each channel in the network model has a width of 36 bits. Data packets have a fixed size of 180 bits, thus they are separated into 5 flits at sender side. According to $h_{max} = 3$, the model features four VCs for data communication and one additional VC for control flits. All network switches are input buffered. We have simulated our network for 50 kcycles in saturation mode using uniform random traffic. The maximum throughput (TP $[\frac{received\ flits}{nodes \cdot cycles}]$) for each hierarchical network configuration is shown in Table 2.

The throughput of the flat network is 0.027 flits per node per cycle. When introducing hierarchy to the network, the throughput is increased for all configurations except for (4 × 2,1 × 2). For (4 × 2,1 × 2) the throughput is comparable to the flat network. The higher throughput can be attributed to the *Up* to *Down* turn allowed in each hierarchical unit. Because of the additional turns, the routing is less restrictive than *Up/Down* routing applied to the flat network.

Please cite this article as: G. Schley et al., Reconfigurable fault tolerant routing for networks-on-chip with logical hierarchy, Computers and Electrical Engineering (2016), http://dx.doi.org/10.1016/j.compeleceng.2016.02.013

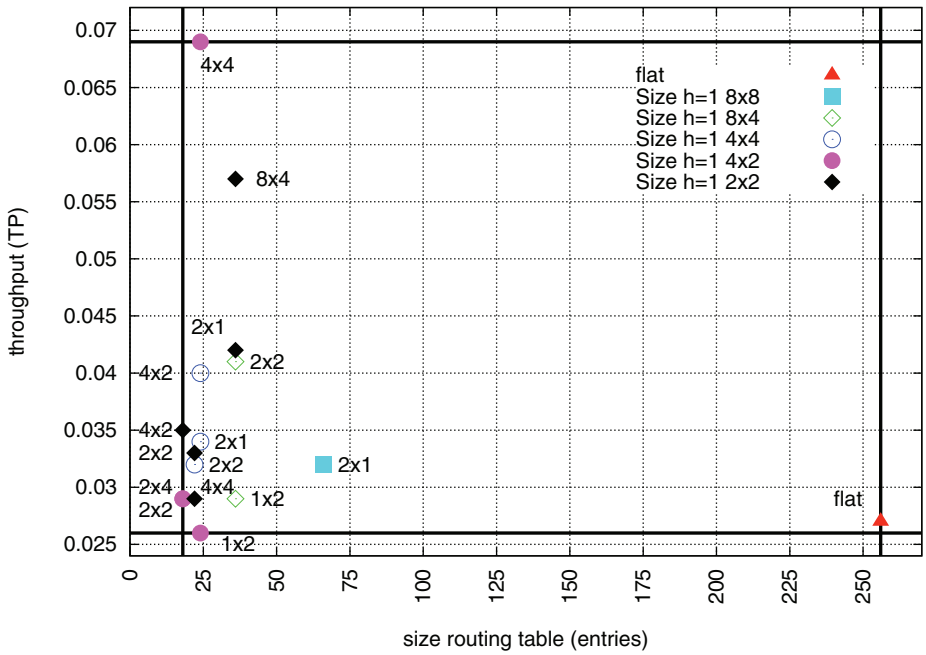

Fig. 7. Design space.

Table size and throughput have an impact on the scalability property of our routing and its performance. The resulting design space spanned by table size and throughput results is shown in Fig. 7. The different configurations lead to different implementation points in the design space. A configuration is considered to be better than others, if it requires fewer routing table entries and provides higher throughput. The optimum is situated in the upper left corner of the design space. According to the design space, configuration (4×2,4×4) and (2×2,4×2) are pareto optimal for a network size of 16×16. Beside the two pareto optimal configurations, configuration (2×2,8×4) provides also a good tradeoff between number of routing table entries (36 entries) and throughput (0.057 flits/node/cycle).

7.3. Reconfiguration performance

We now consider the required time for routing reconfiguration in case of permanent faults. For online routing reconfiguration, the reconfiguration process should be completed as fast as possible so that the NoC can resume normal operation. In a second simulation, we have measured the required reconfiguration time for the flat network as well as for $h_{max} = 3$ levels and have compared the results to the distributed reconfiguration approach *Ariadne* [10]. Simulations have shown that the time required for reconfiguration is mainly composed of the time required for:

1. Updating the *ETG*.
2. Computation of valid paths, and
3. generation of routing table entries.

For the hierarchical network configurations, the reconfiguration time further depends on whether only the routing within a level 1 unit has to be adapted or if routing on level 2 has to be changed. If the routing has to be changed on level 2, the total reconfiguration time is composed of the time required for updating the *ETG* on level 2 and the time for reconfiguration on level 1. The time for updating the *ETG* on level 2 is omitted if the routing only has to be adapted on level 1. The portion of the total required reconfiguration time for the flat and hierarchical networks is shown in Fig. 8.

In the case of the flat network, the required time for reconfiguration is more than 160 kcycles. According to [10], *Ariadne* requires $|N|^2$ cycles for routing reconfiguration and thus approximately 65 kcycles are required for a 16×16 network (indicated by the horizontal line in Fig. 8). This implies that for the flat network the reconfiguration time of our approach is about 2.5 times higher compared to *Ariadne*. This is mainly caused by the time required for path computation. However, as it can be seen in Fig. 8, the required reconfiguration time for all hierarchical network configurations is smaller than 65 kcycles required by *Ariadne*. This is even the case if the *ETG* has to be updated on level 2. The reason for this reduction in time is the reduced number of vertices in the *ETG* (cf. *Entries* in Table 2). The minimum total reconfiguration time of approximately 20 kcycles is obtained for configuration (4×4,8×8). Only less than 11 kcycles are required for configuration (2×2,8×4) if the routing has only to be adapted on level 1.

Taking into account both the results in the design space and the reconfiguration time, the configuration (4×2,4×4) is the most suitable for a 16×16 NoC topology. While for this configuration the throughput increases about 2.5 times com-

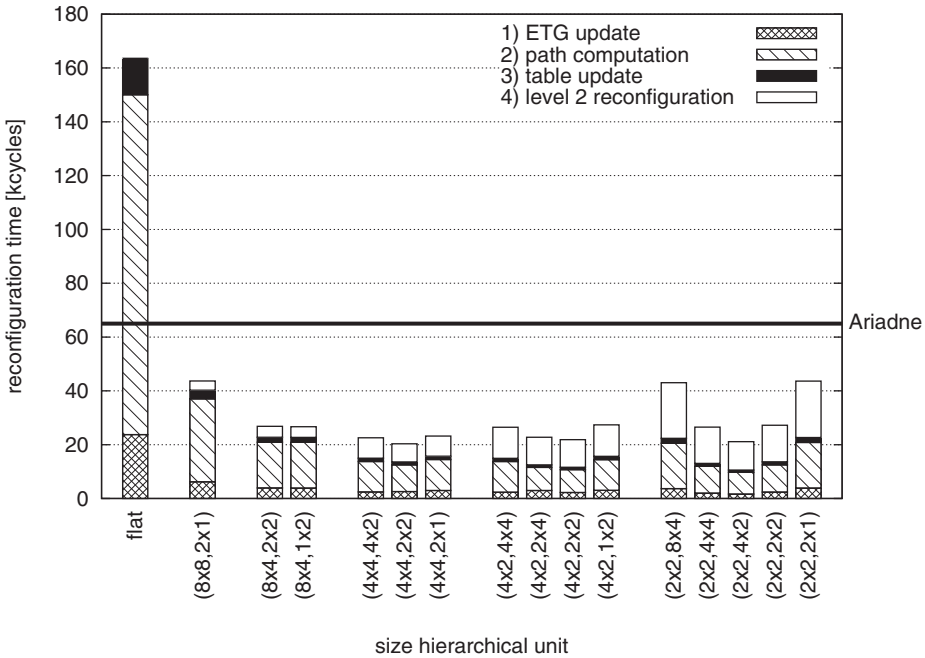

Fig. 8. Required time for routing reconfiguration.

Table 3
Reconfiguration success rate with different bit error probabilities.

Bit error (10^{-3})	3	5	7	9	11	15	20	25	30
Success rate	1	1	1	0.8	0.8	0.5	0.3	0.2	0

pared to the flat network, at the same time the number of table entries decreases about a factor of seven and the required reconfiguration time decreases about a factor of 6. Compared to *Ariadne* the reconfiguration time is four times smaller if routing is adapted on level 1. If the *ETG* has to be updated on level 2, the required reconfiguration time is approximately 41% compared to *Ariadne*.

Finally, we have analyzed the rate for successful routing table reconfiguration. Transient faults may corrupt reconfiguration flits sent to the REU. To investigate their impact on the success rate we have used different error rates for transient faults. For this purpose, we have written a Matlab script to model an error detection unit (CRC) and to inject random errors with different bit error probabilities to reconfiguration flits and retransmission request flits. The experiment is repeated for nine different bit error probabilities. Table 3 shows the success rate of local reconfiguration for each bit error probability. Note that the usual bit error probabilities caused by transient faults are in the range of 10^{-9}–10^{-20} [23]. However, higher values are used here for evaluation purposes.

The results in Table 3 show that for a bit error probability up to 0.007, the REU was always able to reconfigure the routing table successfully. Moreover, even for high bit error probabilities, up to 0.011, the table was reconfigured in at least 80% of the cases. For error probabilities higher than 0.011, the success rate shows a strong decrease due to the increasing number of corrupted reconfiguration flits and retransmission request flits. As error probabilities are much smaller than the one used for evaluation, the REU guarantees successful reconfiguration in case of realistic error probabilities for transient faults.

8. Conclusion

Adding logical hierarchy to Networks-on-Chip (NoCs) offers significant benefits compared to NoCs with flat organization. In particular, logical hierarchy makes routing tables a feasible design choice. This is achieved by having full table entries only for nodes in the same logical network unit, and by merging routing information for other nodes through hierarchical abstraction. Thereby, a routing table for a switch in a 256 node NoC requires only less than 20% of the switch's chip area, whereas a fully implemented table for a flat NoC would double the switch's area. Furthermore, evaluation results for a 16×16 mesh topology with three hierarchy levels show that the data throughput is doubled compared to a non-hierarchical network topology.

Please cite this article as: G. Schley et al., Reconfigurable fault tolerant routing for networks-on-chip with logical hierarchy, Computers and Electrical Engineering (2016), http://dx.doi.org/10.1016/j.compeleceng.2016.02.013

Hierarchically organized routing tables can then be used to implement highly flexible and controlled fault tolerance. This is done by capturing the topology of the partly-faulty network in an enhanced topology graph, by computing new routes with a modified Dijkstra algorithm that takes a deadlock-free baseline routing into account, and by updating the routing tables in a fault-tolerant way, as described in this paper. Per switch, the additional components required for fault tolerance occupy only 6% of the switch's chip area. Thanks to the logical NoC hierarchy, and with proper choices for its organization, reconfiguration time can be reduced to less than one third, compared to the state-of-the-art approach of the Ariadne network. Moreover, our suggested routing table reconfiguration protocol is extremely robust against transmission errors, offering correct reconfiguration even if the bit error probability is in the order of 10^{-3}.

Acknowledgment

This work has been supported by the German Research Foundation (Deutsche Forschungsgemeinschaft - DFG) under grant Ra 1889/4-1.

References

[1] Intel Xeon Phi Coprocessor. visited April 2015. https://software.intel.com/de-de/mic-developer.
[2] Tilera Tile-MX Multicore Processor. visited April 2015. http://www.tilera.com/.
[3] International Technology Roadmap For Semiconductors. visited April 2015. http://www.itrs.net/.
[4] Borkar S. Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. IEEE Micro 2005;25(6):10–16. doi:10.1109/MM.2005.110.
[5] Lienig J. Electromigration and its impact on physical design in future technologies. In: Proceedings of ACM international symposium on physical design (ISPD); 2013. p. 33–40. ISBN 978-1-4503-1954-6. doi:10.1145/2451916.2451925.
[6] Schroeder M, Birrell A, Burrows M, Murray H, Needham R, Rodeheffer T, et al. Autonet: a high-speed, self-configuring local area network using point-to-point links. IEEE Journal on Selected Areas in Communications 1991;9(8):1318–35. doi:10.1109/49.105178.
[7] Schley G, Radetzki M. Fault tolerant routing for hierarchically organized networks-on-chip. In: Proceedings of 23rd euromicro international conference on parallel, distributed and network-based processing (PDP); 2015. p. 379–86. doi:10.1109/PDP.2015.36.
[8] Wachter E, Erichsen A, Amory A, Moraes F. Topology-agnostic fault-tolerant NoC routing method. In: Proceedings of design, automation & test in Europe conference (DATE); 2013. p. 1595–600. doi:10.7873/DATE.2013.324.
[9] Robles-Gomez A, Bermudez A, Casado R. A deadlock-free dynamic reconfiguration scheme for source routing networks using close up∗/down∗ graphs. IEEE Trans Parallel Distrib Syst (TPDS) 2011;22(10):1641–52. doi:10.1109/TPDS.2011.79.
[10] Aisopos K, DeOrio A, Peh L-S, Bertacco V. Ariadne: Agnostic reconfiguration in a disconnected network environment. In: Proceedings of IEEE/ACM international conference on parallel architectures and compilation techniques (PACT); 2011. p. 298–309. doi:10.1109/PACT.2011.61.
[11] 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: Proceedings of the 20th international parallel and distributed processing symposium (IPDPS); 2006. p. 105–14. doi:10.1109/IPDPS.2006.1639341.
[12] Lee D, Parikh R, Bertacco V. Brisk and limited-impact NoC routing reconfiguration. In: Proceedings of the conference on design, automation & test in Europe (DATE); 2014. p. 1–6. ISBN 978-3-9815370-2-4. doi:10.7873/DATE.2014.319.
[13] Ebrahimi M, Wang J, Huang L, Daneshtalab M, Jantsch A. Rescuing healthy cores against disabled routers. In: Proceedings of IEEE international symposium on defect and fault tolerance in VLSI and nanotechnology systems (DFT); 2014. p. 98–103. doi:10.1109/DFT.2014.6962086.
[14] Rodrigo S, Flich J, Roca A, Medardoni S, Bertozzi D, Camacho J, et al. Addressing manufacturing challenges with cost-efficient fault tolerant routing. In: Proceedings of 4th IEEE international symposium on networks-on-chip (NOCS); 2010. p. 25–32. doi:10.1109/NOCS.2010.12.
[15] Jouraku A, Koibuchi M, Amano H. An effective design of deadlock-free routing algorithms based on 2d turn model for irregular networks. IEEE Trans Parallel Distrib Syst (TPDS) 2007;18(3):320–33. doi:10.1109/TPDS.2007.36.
[16] Sancho JC, Robles A, Duato J. An effective methodology to improve the performance of the up∗/down∗ routing algorithm. IEEE Trans Parallel Distrib Syst (TPDS) 2004;15(8):740–54. doi:10.1109/TPDS.2004.28.
[17] Xu TC, Schley G, Liljeberg P, Radetzki M, Plosila J, Tenhunen H. Optimal placement of vertical connections in 3d network-on-chip. J Syst Arch 2013;59(7):441–54. doi:10.1016/j.sysarc.2013.05.002.
[18] Kakoee M, Bertacco V, Benini L. A distributed and topology-agnostic approach for on-line NoC testing. In: Proceedings of 5th IEEE/ACM international symposium on networks on chip (NOCS); 2011. p. 113–20. doi:10.1145/1999946.1999965.
[19] Grecu C, Ivanov A, Saleh R, Sogomonyan E, Pande PP. On-line fault detection and location for NoC interconnects. In: Proceedings of 12th ieee international on-line testing symposium (IOLTS); 2006. p. 145–50. doi:10.1109/IOLTS.2006.44.
[20] Alaghi A, Karimi N, Sedghi M, Navabi Z. Online noc switch fault detection and diagnosis using a high level fault model. In: Proceedings of 22nd IEEE international symposium on defect and fault-tolerance in VLSI systems (DFT); 2007. p. 21–9. doi:10.1109/DFT.2007.55.
[21] Si2 (Silicon Integration Initiative). visited April 2015. http://www.si2.org/openeda.si2.org/projects/nangatelib.
[22] HP Labs CACTI. visited April 2015. http://www.hpl.hp.com/research/cacti/.
[23] Ali M, Welzl M, Hessler S. A fault tolerant mechanism for handling permanent and transient failures in a network on chip. In: Proceedings of 4th international conference on information technology (ITNG); 2007. p. 1027–32. doi:10.1109/ITNG.2007.5.

Gert Schley received the Dipl-Ing (FH) degree in electrical engineering and information technologies in 2007 and the M.S. degree in embedded systems engineering in 2009 from the University of Applied Sciences, Pforzheim, Germany. Since 2009, he has been a Research Scientist with the Embedded Systems Group, University of Stuttgart. His research interests include hierarchical architectures and cross-layer fault tolerance for Network-on-Chip.

Ibrahim Ahmed received his Bachelor of Science in electronics engineering from the German University in Cairo in 2012. He received his Masters of Science in information technology and embedded systems from the University of Stuttgart in 2014. Since 2015, he has been a PhD student at University of Toronto. His research interests include FPGA architecture, VLSI and computer architecture.

Muhammad Afzal received B.Sc Electrical Engineering from AJK University - Pakistan and M.Sc Embedded Systems Engineering from University of Stuttgart - Germany, in 2009 and 2014, respectively. He contributed in research work related to reconfigurable NoC switch in collaboration with Embedded Systems Group, University of Stuttgart. Since 2014 he has been working as an Application Engineer in Altium Europe GmbH.

Martin Radetzki is Professor of Embedded Systems Engineering with the University of Stuttgart. He received the Dipl.-Inform. and Dr.-Ing. degrees from the University of Oldenburg, Germany, in 1996 and 2000, respectively. His research interests include modelling and parallel simulation of embedded systems, design of robust systems, and architecture of fault-tolerant networks-on-chip.