Algorithm for the Choice of Topology in Reconfigurable On-Chip Networks with Real-Time Support

Kristina Kunert
Centre for Research on Embedded Systems (CERES), Halmstad University, Sweden
kristina.kunert@ide.hh.se

Mattias Weckstén
Centre for Research on Embedded Systems (CERES), Halmstad University, Sweden
mattias.wecksten@ide.hh.se

Magnus Jonsson
Centre for Research on Embedded Systems (CERES), Halmstad University, Sweden
magnus.jonsson@ide.hh.se

ABSTRACT
Many future embedded systems are likely to contain System-on-Chip solutions with on-chip networks and in order to achieve high aggregated throughputs in these networks, a switched topology can be used. For further performance improvements, the topology can be adapted to application demands, either when designing the chip or by run-time reconfiguration between different predefined application modes. In this paper, we present an algorithm for the choice of topology in, e.g., on-chip networks, considering real-time demands in terms of throughput and delay often put on such systems. To further address possible real-time demands, we include a feasibility analysis to check that the application, when mapped onto the system, will behave in line with its real-time demands. With input information about traffic characteristics, our algorithm creates a topology and generates routing information for all logical traffic channels. In a case study, we show that our algorithm results in a topology that can outperform the use of state of the art topologies for high-performance computer architectures.

Categories and Subject Descriptors
C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design – *distributed networks, network communications, network topology, packet-switching networks.*

General Terms
Algorithms, Performance, Design.

Keywords
Network-on-Chip, topology design, feasibility analysis, real-time communication, reconfigurable systems.

Nano-Net 2007 September 24-26, 2007, Catania, Italy.
 ISBN 978-963-9799-10-3
DOI 10.4108/ICST.NANONET2007.2148

1. INTRODUCTION
Today, more and more embedded applications have real-time requirements and to be able to fulfil these requirements, networks with real-time support are needed. For applications such as radio base stations and radar signal processing, not only high bandwidth is required, but also deterministic real-time properties of the communication are crucial. In the future, we see the possibility of chips with tens or even hundreds of processors connected by a switched network, running applications with real-time requirements. The advantages of using reconfigurability in a Network-on-Chip (NoC) are obvious. The topology can be adapted to different applications, different modes of applications, or even different traffic patterns in one single application. In this way, the high cost of designing and manufacturing a new chip can be reduced.

This paper presents an algorithm for the design of network topologies in, e.g., packet-switched on-chip networks [2], considering the real-time demands in terms of throughput and delay often put on such systems. To further address those real-time demands, we include a feasibility analysis to check that the application, when mapped onto the system, will behave in line with its real-time demands. With input information about the traffic characteristics, our algorithm creates a topology and generates routing information for all logical real-time channels. The proposed algorithm can be used both to select topologies to reconfigure between in case hardware for this is provided, and in the design stage of a static NoC for the decision on which network topology to implement for any intended target application. When having a reconfigurable topology (e.g., implemented by a crossbar), a specific topology can be chosen in the design stage for each working mode executed during run-time. A case study with simulation of two types of radar signal processing chains demonstrates the algorithm's applicability for applications with different traffic patterns.

The problem of topology design or link allocation according to certain parameters is found in a variety of research areas. Most research on topology design today is conducted in the area of System-on-Chip (SoC) design and NoC architectures. At a lower level, the advance of research on on-chip interconnection networks is mainly driven by the limitations of the interconnections used today, i.e., principally bus networks. Busses are simple to implement but have scalability problems since they can only transfer one message at a time. The authors of [3] and [5] point out the advantage of designing arbitrary topologies, adapted towards a target application's specific demands and properties. In [5], a recursive algorithm is suggested to both find minimal topologies and share the communication medium with low

contention. However, those papers do not present algorithms considering real-time demands. SUNMAP [1,10,11] is a mapping algorithm which can map any target application on a certain limited set of standard topologies, considering both parameters given by the technology and communication requirements. The algorithm we propose in this paper contributes by finding a suitable topology as well as a traffic mapping and schedule. Moreover, we do not restrict ourselves to a fixed set of topologies. The authors of [14] use linear programming based techniques to design application specific NoC architectures, but in contrast to our approach, no delay-bound guarantees for real-time traffic are provided. Since scheduling and topology generation are known to be NP-complete problems, we have to resort to a heuristic solution. A similar problem, finding a logical topology to meet all existing traffic demands in an optical network, is solved in [7] by the use of a shortest-path algorithm that constructs a route for one light path (a virtual channel comprising one single wavelength between two nodes) at a time. Although not the same problem, this method resembles our own solution. However, the major difference, again, is that our solution takes the real-time demands of the traffic into account. In [4] the authors describe a routing algorithm very similar to the one presented in this paper, a shortest path algorithm, where path selection is based on the timing requirements. To guarantee the real-time behaviour of the resulting system the authors use a TDMA (time-division multiple access) scheme. The experiments presented in [4] are based on silicon level simulation, resulting in area and energy parameters. However, although mapping and routing problems are addressed in the paper, topology exploration is not. In [15] the authors use a similar shortest path algorithm for routing but in this case the path selection is not based on timing requirements directly, but rather on energy consumption and operating frequency of the links. The experiments presented in [15] are also based on silicon level simulation. Although we decided not to conduct experiments on silicone level, the results of our real-time performance oriented work could be used for guidance of low-level implementation.

The rest of the paper is organized as follows. In Section 2 we describe our system's architecture, while Section 3 introduces our topology choice algorithm. Section 4 describes the feasibility analysis as used in the algorithm. A case study is implemented to evaluate the algorithm and the results are discussed in Section 5. Conclusions are drawn and possibilities for future work are pointed out in Section 6.

2. SYSTEM ARCHITECTURE

Our target system architecture consists of a reconfigurable topology, connecting M switched processor clusters, each containing N communicating end nodes (see Figure 1). The incorporation of the reconfigurable interconnection network allows for the adaptation to different application characteristics and can therefore improve network performance. While the links between the end nodes and the router in a cluster are bidirectional, the links between the router and the reconfigurable topology are assumed to be unidirectional with identical amounts of bandwidth. Using unidirectional links makes it possible to utilize the bandwidth more efficiently as the flexibility in traffic allocation increases. Communication between nodes in the same cluster is assumed to be unrelated to the cluster-to-cluster communication, as intra-cluster communication has no influence on the bandwidth bottleneck over the reconfigurable topology. This system model

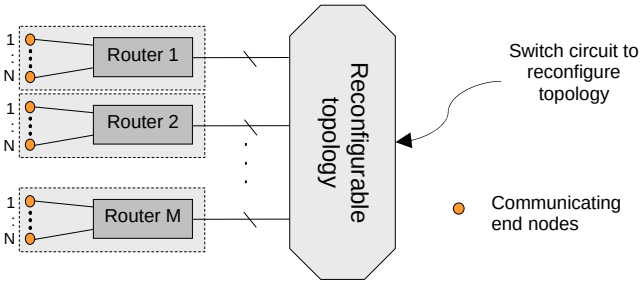


Figure 1. Schematic system architecture.

can be mapped onto different kinds of networks on different scales, not only NoCs. It can also encompass networks between nodes on several boards in one rack or between boards in several racks.

3. TOPOLOGY SELECTION ALGORITHM

This section provides the assumptions for the algorithm, and gives a detailed description of its different stages. At this point of time, computational complexity is not the major limitation, as the algorithm can be run offline for all possible online working modes. The results of the algorithm are the topologies of the interconnection network that the reconfigurable system will switch between during run-time and the necessary routing information. For a static NoC, the algorithm is just executed once, at design stage, to get the topology for the intended application. Due to the limited amount of space, we refer to [8] for a more detailed specification of the algorithm.

3.1 Assumptions

For each target application, the algorithm uses input traffic specifications in the form of real-time channels. A real-time channel is a logical flow, modelled as a virtual unidirectional channel with the following parameters (for any real-time channel i): the source S_i , the destination D_i , the maximum message length in bits (per period $T_{P,i}$) C_i , the deadline in seconds $T_{D,i}$, and the period (maximum message inter-arrival time) in seconds $T_{P,i}$. Therefore every real-time channel i is characterized in the following form:

$$CH_i = \{S_i, D_i, C_i, T_{D,i}, T_{P,i}\} \quad (1)$$

Additionally, the throughput demand in bits per second of each real-time channel i is given by:

$$B_i = \frac{C_i}{T_{P,i}} \quad (2)$$

Also provided is the number of input and output ports on each router towards the topology, denoted as *MAX_LINK*. The reconfigurable topology can connect each such router output port towards the topology to an arbitrary free input port on another router. Assuming priority support in the routers, non real-time traffic can be given lower priority and will interfere with the real-time traffic. However, this interference will be bounded to a single packet due to the non-preemptiveness of a packet.

For the intra-cluster traffic from the end nodes towards the router, which they are directly connected to, and vice versa, simply the link utilization is checked to ensure that there are no bottlenecks at

this first and last hop of the communication path. However, they will still have to be included in the delay analysis, which is described later in the paper (Section 4). After this utilization check, the algorithm will not regard the links between end nodes and routers, but only the links over the reconfigurable topology.

3.2 Description

The design approach of the algorithm has been to prioritize the allocation of physical links to connect pairs of nodes with real-time channels between each other, demanding high guaranteed throughput and/or short bounded delays. When multihop communication over the reconfigurable topology is needed, we prioritize shorter paths for this throughput and/or delay demanding communication. The algorithm works in three main phases: singlehop routing with link allocation, multihop routing with link allocation, and feasibility testing. Prior to the main phases, certain initializations are needed. In order to decide upon the sequence in which the logical real-time channels should be allocated, we introduce an individual weight W_i for each channel *i*. This weight is calculated by one of two weight functions, depending on the phase in which the channel is allocated. Details on those weight functions are given in the detailed description of the different phases below.

1) *Phase 0 - Initializations.* For the algorithm to know in which order to process the real-time channel demands during the upcoming Phase 1, the initialization phase is started by calculating W_i for each channel *i*. A channel's weight in Phase 1 is dependent on its throughput demand and its deadline, i.e. C_i, $T_{D,i}$, and $T_{P,i}$. Furthermore, this first weight function is solely aimed towards the singlehop routing case, where the length of a routing path is equal to three for every path. "Singlehop" in this context denotes thus only the hop over the reconfigurable topology. Considering these facts, we define the following weight function *s_wfcn* for Phase 1:

$$W_i = s\_wfcn(C_i, T_{D,i}, T_{P,i}) = \frac{\frac{C_i}{T_{P,i}}}{MIN\left(1, \frac{T_{D,i}}{3 \cdot T_{P,i}}\right)} \quad (3)$$

The minimum value function used in the denominator leads to one of two cases. If the deadline $T_{D,i}$ is longer than three times the period $T_{P,i}$ (due to a path length of three and the assumption of equal deadline partitioning; more on deadline partitioning in Section 4.), then the only relevant factor for the weight of the link becomes its throughput demand (the numerator). On the other hand, if the deadline becomes shorter than three times the period, then these two parameters become significant enough to be included in the weight calculation. In other words, the hop through the topology is assumed to be either utilization constrained or delay constrained. This is an approximating hypothesis based on the fact that, for the case when the period is equal or shorter to the per-hop deadline for all channels, only the aggregated utilization needs to be verified to be less or equal to 100% [9]. Those results have their origin from the field of processor task scheduling.

After having assigned an individual weight to each logical real-time channel, the channels are grouped into bundles, each bundle $G_{S,D}$ containing channels with a specific source-destination pair *(S,D)*. The channels in each bundle are sorted in a priority queue $Q_{S,D}$, with the channel that has the highest bandwidth demand getting the highest priority. By summarizing the individual weights W_i of all channels in one bundle, each bundle is assigned an individual weight $W_{S,D}$. Lastly, all bundles are sorted in a priority queue Q_{bundle}, giving highest priority to the bundle with maximum weight.

2) *Phase 1 - Singlehop routing with link allocation.* After having initiated the necessary parameters in Phase 0, the algorithm enters Phase 1, during which the algorithm routes singlehop paths, and when necessary allocates new physical links over the topology. The algorithm sequentially tries to allocate all logical real-time channels contained in bundle *G* (defined to be the one with the highest weight) through direct links over the topology. The channels are treated according to bandwidth demand, with the highest demand getting the highest priority. After having allocated all channels in one bundle, the algorithm treats the bundle with the next highest weight in the same fashion.

When allocating a path for a channel, the algorithm first examines if a link already exists between the channel's source and destination router and if its capacity can meet the throughput demands of the channel. The sum of the throughput demands of all logical real-time channels over the same physical link must not exceed the maximum bit rate of that link. If it does not, the additional channel can be allocated on the existing link. (More details about the utilization constraint will follow in Section 4.) In case there is not enough capacity left on the link, the algorithm checks for the possibility to allocate a new physical link, parallel to the recently examined, using unallocated output ports at the source router and unallocated input ports at the destination router. When the algorithm no longer can find a singlehop path for a channel, it only tries to allocate the remaining channels in the current bundle as singlehop paths before terminating Phase 1. Then it will enter into Phase 2 to start with multihop routing. In case all real-time channels could be allocated in Phase 1, the algorithm continues directly with the feasibility check in Phase 3.

3) *Phase 2 - Multihop routing with link allocation.* In the multihop routing phase, the prioritization is solely based on the deadlines specified in the real-time channel demands and the concept of bundles is no longer used. The decision of making the priority solely dependent upon the end-to-end deadline of the real-time channel is based on the approximate assumption of the uniformity of all links, resulting into longer delays over paths with a larger number of hops. The new individual weight W_i of all remaining channels is calculated by a new weight function *m_wfcn*, which has $T_{D,i}$ as its only input parameter:

$$W_i = m\_wfcn(T_{D,i}) = \frac{1}{T_{D,i}} \quad (4)$$

All individual channels are sorted in a priority queue $Q_{channel}$, with the highest weight and, by that, the shortest relative deadline getting highest priority. Each routing path *R* is found by an unweighted shortest path algorithm, related to Dijkstra's algorithm, with all link costs being one. This leads to a routing algorithm with a hop-based cost metric, and therefore the shortest path signifies the path with the smallest number of hops between S_i and D_i. For each real-time channel demand, the shortest path algorithm searches a route through the partly allocated network, trying primarily to use the existing links unless their remaining capacity is too small to cope with the throughput demand of the channel currently under consideration. Only secondarily new, unallocated, links are set up. In case a path could not be found for all channels, the algorithm is terminated, as no suitable topology could be found for the given traffic specifications, otherwise the algorithm proceeds further to the feasibility check (Phase 3).

4) *Phase 3 - Feasibility testing*. A feasibility test has been added to verify that all real-time demands can be met. In case of a positive outcome, our algorithm provides as output the recommended network topology and a routing table belonging to the traffic demands. Otherwise, no suitable topology for the target application could be found. The theory of the feasibility analysis and the details of its application in our algorithm are given in the next section.

4. FEASIBILITY ANALYSIS

To be able to determine the performance characteristics for hard real-time traffic over an arbitrary topology, this section provides a throughput guarantee and delay bound analysis. Due to the assumption of earliest deadline first (EDF) scheduling in all end nodes and intermediate routers, the feasibility analysis suggested in [6] can be used in a similar manner for our network. Real-time channels correspond to periodic tasks, where a physical link in the network can be seen as a processor and the maximum message length in our traffic specification is the equivalent to the worst-case execution time C_i for task i in the original analysis. For the description of the feasibility check, a number of concepts need to be defined.

- The utilization U of periodic real-time traffic is defined as

$$U = \sum_i \left(\frac{C_i}{T_{P,i}} \right) \tag{5}$$

  where $T_{P,i}$ denotes the period (minimum message inter-arrival time) of traffic over the logical channel i.
- The hyperperiod *HP* is the least common multiple of all periods of a periodic task set, i.e., the length of time from when all tasks' periods start at the same time, until they start at the same time again.
- The busyperiod *BP* is any interval within *HP* in which the resource, in our case the link, is not idle.
- The point in time t signifies the number of time slots elapsed since the beginning of the *HP*.
- The traffic demand on the network corresponds to the processor demand in a real-time system and is defined by the workload function *h(t)*. *h(t)* is calculated as the sum of C_i for all message instances of all real-time channels with an absolute deadline less then or equal to t. *h(t)* is computed as follows [13].

$$h(t) = \sum_{T_{D,i} \le t} \left(1 + \left\lfloor \frac{(t - T_{D,i})}{T_{P,i}} \right\rfloor \right) \cdot C_i \tag{6}$$

In our algorithm, the feasibility analysis is conducted for each physical link, which means that the total end-to-end deadline has to be partitioned into local single hop deadlines instead. For the matter of simplicity we have chosen an even distribution of the deadline over the entire path, i.e., the local deadline d_i corresponds to the end-to-end delay bound $T_{D,i}$ divided by the number of hops NoH_j that the relevant path j consists of.

$$d_i = \frac{T_{D,i}}{NoH_j} \tag{7}$$

However, this type of feasibility analysis assumes fully pre-emptive tasks. In interconnection networks packets normally cannot be pre-empted and consequently the possibility of further delay has to be considered. Therefore, we define the blocking time *BT*, which denotes the maximum blocking time that one (possibly lower-priority) packet can introduce to the system, i.e., *BT* equals the transmission time of a maximum size packet. This compensation results in a further shortening of the local delay bound.

$$d_i' = d_i - BT \tag{8}$$

This means that the workload function is remodelled as follows.

$$h(t) = \sum_{d_i \le t} \left(1 + \left\lfloor \frac{(t - d_i')}{T_{P,i}} \right\rfloor \right) \cdot C_i \tag{9}$$

While keeping $U \le 1$ is a necessary condition for being able to feasibly schedule periodic tasks by EDF [13], a second constraint was introduced in [13] to ensure the feasibility of the system when adding a new task or channel:

$$h(t) \le t \quad \forall t \tag{10}$$

This restriction introduces a high computational complexity, but the number of instances of evaluation can be reduced to the number of integer time values during an interval upper bounded by BP_1, the first *BP* in the first *HP* of the schedule where all periods start at time zero. Again, this has to be adapted to our hop-by-hop calculations by exchanging the end-to-end delay bound $T_{D,i}$ with the local delay bound d_i. In our case of non-pre-emptive communication, the local delay bound, d_i, further has to be exchanged for d_i' because of the possibility of blockage, which results in the following instances of t having to be checked:

$$t \in \bigcup_{i=1} \{ m \cdot T_{P,i} + d_i' : m = 0,1,2... \} \tag{11}$$

where

$$t \in [1; BP_1] \tag{12}$$

In order for the whole path to be accepted, all contained hops must be found feasible.

5. CASE STUDY

The topologies generated by the proposed algorithm are evaluated by comparing the network efficiency of the generated topologies with that of a 2D-torus, a standard topology for high-performance computer architectures. In a 2D-torus each node has four neighbours and the topology has wrap-around edges. The efficiency of a given topology is defined as the pair (U_{net} $|L|$), where U_{net} denotes the total network utilization, or the sum of the utilization U_i for all links i, and $|L|$ is the number of physical links used, or the sum of the ceiling function of the utilization U_i for all links i.

$$U_{net} = \sum_i U_i \tag{13}$$

$$|L| = \sum_i \lceil U_i \rceil \tag{14}$$

A lower value of U_{net} signifies a more energy efficient topology since it consumes less of the total network resources. In cases of similar values of U_{net}, a lower value of $|L|$ indicates a more

hardware efficient topology, since a smaller number of links is needed as the capacity of the existing links is used to a higher degree. Even though it can be argued that it is better to use all available resources, a larger number of links with lower utilization could be an indication that the algorithm has chosen longer paths than necessary. In other words, a low value of $|L|$ indicates an efficient algorithm that can lead to a higher possible amount of guaranteed real-time traffic in case this is requested.

In order to demonstrate the algorithm's applicability for use cases with different traffic patterns, the algorithm has been tested with two types of radar signal processing chains. Case 1 is dominated by one-to-many and many-to-one communication, while Case 2 contains more pipelined traffic and just a minor amount of one-to-many and many-to-one transmissions. A specification of the traffic demands is given in Figure 2 and Figure 3. The layout for the 2D-torus was chosen to be 3x4 nodes and the mapping of the nodes and links of the 2D-torus were done manually, keeping $|L|$ at a minimum. In order to be able to compare the efficiency of the

Table1. Total utilization in the network and total utilization of links

		U_{net}*	$\|L\|$**
Case 1: Corner turn	*Generated topology*	*12,4*	*28*
	2D-torus	*21,8*	*32*
Case 2: Pipeline	*Generated topology*	*16*	*19*
	2D-torus	*22.5*	*27*

*Results are given in % of maximum link utilization.
**Results are given in number of used links.

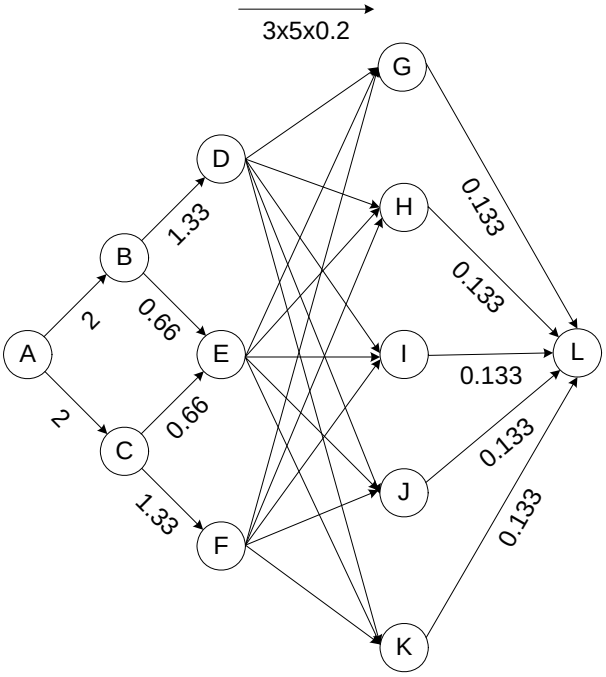


Figure 2. Case 1: Corner turn traffic demands

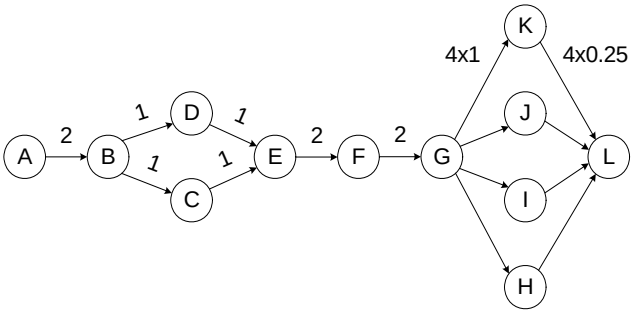


Figure 3. Case 2: Pipeline traffic demands

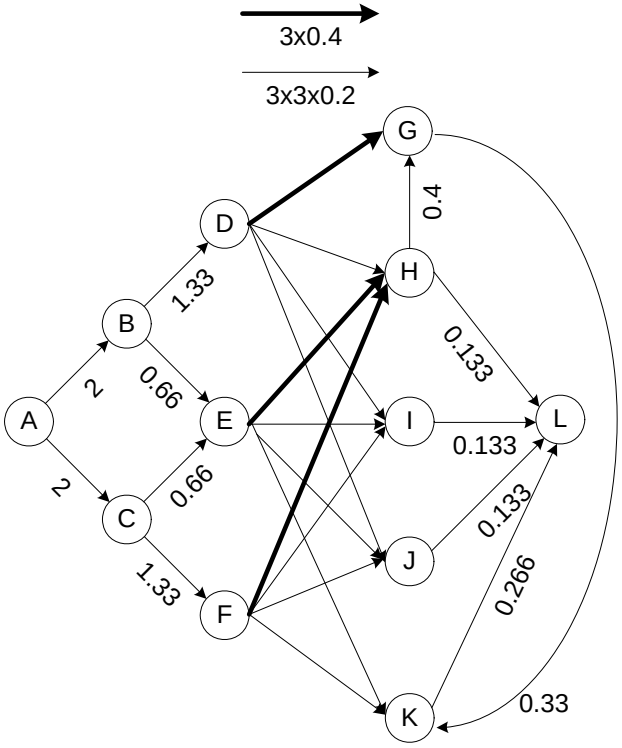


Figure 4. Case 1: Topology and traffic allocation produced as a result of the proposed algorithm

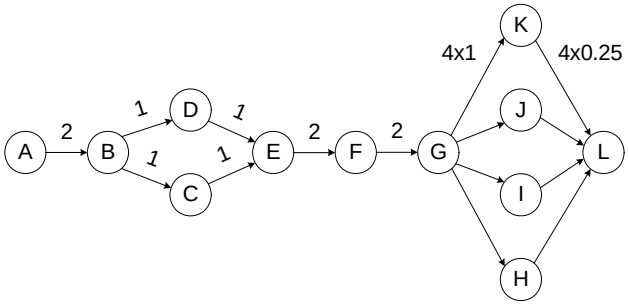
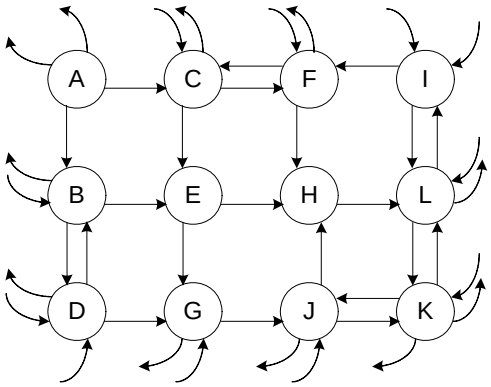


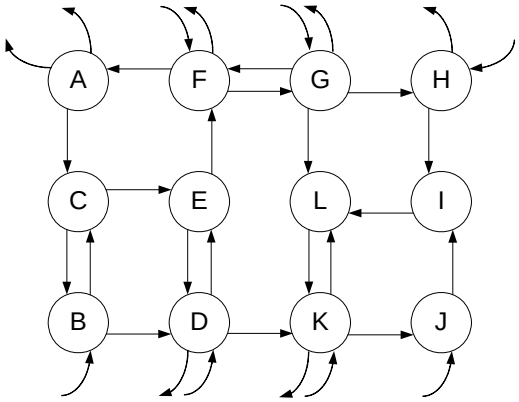
Figure 5. Case 2: Topology and traffic allocation produced as a result of the proposed algorithm



Link	Load	Link	Load
B-E	0.86	H-L	0.666
B-L	0.33	I-F	0.133
C-E	0.66	J-F	0.33
C-F	0.2	J-H	0.2
C-G	0.53	J-K	0.996
D-B	0.2	K-D	0.33
D-K	0.8	K-I	0.2
E-G	0.8	K-J	0.4
E-H	0.4	K-L	0.729
F-H	0.533	L-I	0.4
F-J	0.4	L-K	0.33
G-J	0.863		

All remaining indicated links have a load of 1.

Figure 6. Case 1: Topology and traffic allocation mapped onto a 2D-torus



Link	Load
G-L	0.25
H-J	0.25
I-L	0.75
J-I	0.5
K-L	0.25
K-J	0.25
L-K	0.25

All remaining indicated links have a load of 1.

Figure 7. Case 2: Topology and traffic allocation mapped onto a 2D-torus

generated topologies directly with the 2D-torus, the experiments were carried out with the *MAX_LINK* constraint for the generated topologies being set to four.

The proposed algorithm managed to find feasible solutions within the specified requirements for both cases. As seen in Table 1 and Figures 4-7, the proposed algorithm generated more efficient topologies than the 2D-torus in both cases, and in neither of the cases, the solutions fail the feasibility test. Using the proposed algorithm and a reconfigurable topology, the experiments show a 10-30% reduction of the number of links used and a 20-40% reduction of the used network bandwidth compared to when using a manually configured 2D-torus. Practically this means that the reconfigurable topologies have left more spare resources and can therefore accept a higher amount of real-time traffic in case that was requested. The difference is more noticeable in Case 1 because of the substantial number of one-to-many and many-to-one transmissions. However, even for the pipelined structure of Case 2, the proposed algorithm found a more efficient solution. This depends upon the fact that several transmissions to the same destination in our case require more than one link, and have to be routed over a multi-hop path in a 2D-torus. The reconfigurable topology, however, has the opportunity for the allocation of several parallel links between node pairs. In the case of one-to-many or many-to-one communication where the number of parallel transmissions exceeds four, the usage of multi-hop paths is inevitable though due to the *MAX_LINK* constraint, independent of the topology. However, a reconfigurable topology gives the option of a more flexible path allocation compared to the 2D-torus.

6. CONCLUSION AND FUTURE WORK

It is clear that the SoC/NoC community is in need of more powerful communication networks compared to the commonly used bus hierarchies, both in terms of flexibility and efficiency. This makes a dynamically reconfigurable topology a well-suited alternative. However, to fully utilize the potential of a dynamic topology, efficient tools such as the algorithm presented in this paper, are needed. The advantages of using reconfigurability in a NoC are obvious; the topology can be adapted to different applications, or different modes of applications, and even different traffic patterns in one single application. The proposed topology allocation algorithm has been shown to produce solutions that outperform a manually configured traditional topology for high-performance networks. In comparison with a traditional 2D-torus the results proved that the aggregated network utilization could be decreased with approximately 40% by using the proposed algorithm.

Possible extensions of the current version of the algorithm include taking into account system demands for energy efficiency and fault tolerance. In addition, a more advantageous deadline partitioning could be used to further improve network performance. The next step of the algorithm development is the integration of the feasibility analysis in the actual algorithm and to use this additional information when generating the topology in order to be able to guarantee the timely treatment of hard real-time traffic. Furthermore, there are plans for a more holistic approach to system design.

7. REFERENCES

[1] Bertozzi, D., Jalabert, A., Murali, S., Tamhankar, R., Stergiou, S., Benini, L., and De Micheli, G. NoC Synthesis Flow for Customized Domain Specific Multiprocessor Systems-on-Chip, *IEEE Trans. Parallel and Distr. Syst.*, *16*, 2 (Feb. 2005), 113-129.

[2] Dally, W. J. and Towles, B. Route Packets, Not Wires: On-Chip Interconnection Networks, *38th Conf. on Design Automation. (DAC '01)* (Las Vegas, NV, USA, June 18-22, 2001). 684-689.

[3] García, J. M. and Duato, J. An Algorithm for Dynamic Reconfiguration of a Multicomputer Network, *Proc. 3rd IEEE Symposium on Parallel and Distr. Processing* (Dallas, TX, USA, Dec. 2-5, 1991). 848-855.

[4] Hansson, A., Goossens, K., and Rădulescu, A. A Unified Approach to Constrained Mapping and Routing on Network-on-Chip Architectures, *Proc. 3rd Int. Conf. on Hardware/Software Codesign and System Synthesis (CODES+ISSS'05)* (Jersey City, NJ, USA, Sept. 19-21, 2005). 75-80.

[5] Ho, W. H. and Pinkston, T. M. A Methodology for Designing Efficient On-Chip Interconnects on Well-Behaved Communication Patterns, *Proc. 9th Int. Symposium on High-Performance Computer Architecture. (HPCA-9 '03)* (Anaheim, CA, USA, Feb., 8-12, 2003). 377-388.

[6] Hoang, H. and Jonsson, M. Switched Real-Time Ethernet in Industrial Applications - deadline partitioning, *The 9th Asia-Pacific Conf. on Communication (APCC '03)*, *1* (Penang, Malaysia, Sept. 21-24, 2003). 76-81.

[7] Lee, K. and Shayman, M. Single and Multipath Logical Topology Design and Traffic Grooming Algorithm an IP over WDM Networks, *Proc. 12th Int. Conf. on Computer Communications and Networks (ICCCN '03)* (Dallas, TX, USA, Oct. 20-22, 2003). 59-64

[8] Kunert, K., Weckstén, M., and Jonsson, M., *Algorithm for the Choice of Topology in Reconfigurable Networks with Real-Time Support*, Technical Report IDE0754, School of Information Science, Computer and Electrical Engineering (IDE), Halmstad University, Sweden, 2007.

[9] Liu, C. L. and Layland, J. W. Scheduling Algorithms for Multiprogramming in Hard Real-Time Traffic Environments", *J. Association for Computing Machinery*, *20*, 1, (Jan. 1973).

[10] Murali, S. and De Micheli, G. SUNMAP: a Tool for Automatic Topology Selection and Generation for NoCs", *Proc. 41st Design Automation Conf.* (San Diego, CA, USA, June 7-11, 2004). 914-919.

[11] Murali, S., Benini, L., and de Micheli, G. Mapping and Physical Planning of Networks-on-Chip Architectures with Quality-of-Service Guarantees, *Proc. Asia and South Pacific Design Automation Conference (ASP-DAC '05)*, *1* (Shanghai, China, Jan. 18-21, 2005). 27-32.

[12] Murali, S., Meloni, P., Angiolini, F., Atienza, D., Carta, S., Benini, L., De Micheli, G., and Raffo, L. Designing Application-Specific Networks on Chips with Floorplan Information, *Proc. Int. Conf. on Computer-Aided Design (ICCAD'06)* (San Jose, CA, USA, Nov. 5-9, 2006). 355-362.

[13] Stankovic, J. A., Spuri, M., Ramamritham, K., and Buttazzo, G. C.*Deadline Scheduling for Real-Time Systems - EDF and Related Algorithms*. Kluwer Academic Publishers, Boston, MA, USA, 1998.

[14] Srinivasan, K., Chatha, K. S., and Konjevod, G. Linear Programming Based Techniques for Synthesis of Network-on-Chip Architectures, *Proc. IEEE Int. Conf. on Computer Design: VLSI in Computers & Processors (ICCD '04)* (San Jose, CA, USA, Oct. 11-13, 2004). 422-429.