Packet Routing in Dynamically Changing Networks on Chip

Mateusz Majer, Christophe Bobda, Ali Ahmadinia, Jürgen Teich
Department of Computer Science 12, Hardware-Software-Co-Design
University of Erlangen-Nuremberg, Germany
{majer, bobda, ahmadinia, teich}@cs.fau.de

Abstract

On-line routing strategies for communication in a dynamic network on chip (DyNoC) environment are presented. The DyNoC has been presented as a medium supporting communication among modules which are dynamically placed on a reconfigurable device at run-time. Using simulation, we compare the performance of an adaptive Q-routing algorithm to the well known XY-routing strategy. Both algorithms are adapted to support communication on the DyNoC which is equivalent to routing on meshes with obstacles. In our experiments, Q-routing proves its performance under varying network load while using only local information for its routing decisions.

1. Introduction

Partially reconfigurable devices require efficient algorithms for the placement of tasks onto the device at run-time. For on-line temporal placement, several algorithms have been developed [2, 1, 12, 10]. Apart from the work presented in [1], communication connections among modules are yet not being considered. Although a new placement strategy was developed in [1], which takes the inter-module communication into account, it does not determine how the communication will be realized. Its main advantage is the reduction of communication costs by placing connected modules close to each other. In order to support the dynamic communication which arises when modules are dynamically placed on a device, the reconfigurable device must provide a viable communication infrastructure supporting run-time topology changes. Recently in [5], we presented a new communication infrastructure, the DyNoC (Dynamic Network on Chip). However, packet routing strategies were not part of the investigation. In this paper, we propose and analyze two packet routing strategies for the DyNoC, namely Q-routing and XY-routing. Both are adapted for the

DyNoC. We first motivate the need for adaptive routing protocols by showing that the on-line placement and removal of modules rapidly changes the topology of the communication network.

The rest of the paper is organized as follows: In Section 2, we present previous approaches for handling dynamic on-chip communication. The requirements on the reconfigurable architecture are presented in Section 2.1. Section 2.2 deals with the dynamic connection of placed modules to the network. In Section 3, we propose an extension for XY-routing called S-XY-routing. Section 4 describes the concept of Q-routing for dynamic networks on chip. Section 5 presents our experimental methodology and discusses the simulation results. Finally, Section 6 summarizes the paper and concludes with an outlook.

2. The Dynamic Network on Chip (DyNoC)

Many authors have presented the Network-on-Chip (NoC) concept as a viable and flexible communication solution for a System-on-Chip [7, 8, 3]. NoCs have many advantages (performance, structure and modularity) over global signal wiring. A chip employing a NoC is composed of a set of network clients like DSPs, processors, memory, periphery controllers and custom logic. All implemented network client (NC) physical layouts are constrained to rectangular shapes, also called tiles. Now, instead of connecting these modules using dedicated signal wires, they are connected to a network which routes packets. This network consist of network elements (router nodes) where each network element (NE) is connected to adjacent network elements only, thus forming a two dimensional mesh. While in traditional NoCs fixed locations are defined for processing elements (PEs) as well as for router nodes, our dynamic network on chip approach has much weaker restrictions. Therefore, the DyNoC represents an interesting infrastructure for communication among modules which are dynamically placed at

run-time onto a reconfigurable device.

2.1. Communication infrastructure

With the dynamic on chip packet-based communication, the alteration of the network will not hinder the communication, since packets can always find their way in a strongly connected network. In a static NoC, the clients are placed in rectangular tiles on the chip and communicate with other clients via a fixed network structure. It has been shown in [7] that in this case, the area occupied by the network logic is small (about 6 % of the tile's area).

This ratio is expected to drastically decrease with the rapid logic growth rate observed for reconfigurable devices, e.g. FPGAs. In order to have a better network element to processing element logic area ratio and to make an optimal use of the resources, we impose the following requirements on the communication infrastructure:

- The PEs should be flexible, but coarse grained computing elements. With fine grained PEs, the ratio of network elements to PEs logic area becomes too big which means that the router nodes waste too much device area.

- Each PE should have access to the network. This condition is very important, since it allows any module being placed on the reconfigurable device to access the network via one of its surrounding PEs independent of its placement region.

- PEs should directly communicate with their neighbors. This is helpful because it allows wiring to be done locally within a module boundary.

- Network elements are only connected to adjacent NEs.

- The network logic used for a network element should be flexible enough to be used within the module which is placed over it. Whenever a module is placed in a given region of the device, the network elements inside the module boundary cannot be used for network operations because their channels are deactivated. Therefore, they should be used as additional resources inside the module which covers them. A placed module can use the resources of all the network elements it covers.

Figure 1 shows the communication infrastructure of a reconfigurable device, as described above. In Figure 2 several modules are placed. Obviously, the network becomes dynamic due to placement and relocation of modules at run-time.
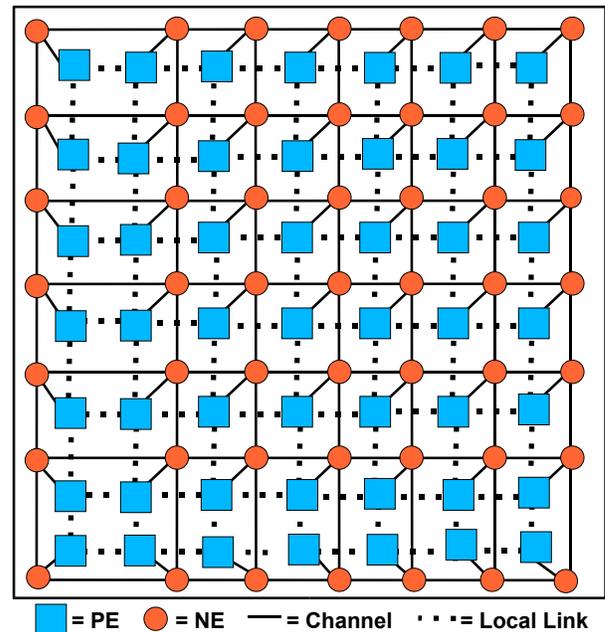

= PE ● **= NE** —— **= Channel** ⋅⋅⋅**= Local Link**

Figure 1. The dynamic NoC approach for reconfigurable communication

Having defined the communication infrastructure of the device, the main questions are to know how to develop modules which can be dynamically connected to the network at run-time. We provide the answers to these questions in the next sections.

2.2. Network access

Each task is implemented as a module, represented by a rectangular tile and stored in a database. A tile encapsulates a circuit implemented with the resources in a given area. Therefore, modules can only access the network using one of the network elements adjacent to its boundary. Without loss of generality, we select a feasible network element on which the upper right PE of the module is attached. After the placement of a new module on the device, the placer will store the coordinates of the feasible network element inside the module as the network address. When placed on the device, modules hide part of the network logic which is restored when they complete their execution. This makes the NoC dynamic. Hence, we call such a network a *dynamic network-on-chip (DyNoC)*.

Because the communication between two modules is established at run-time and since we don't know in advance where modules are going to be placed, we require the cor-
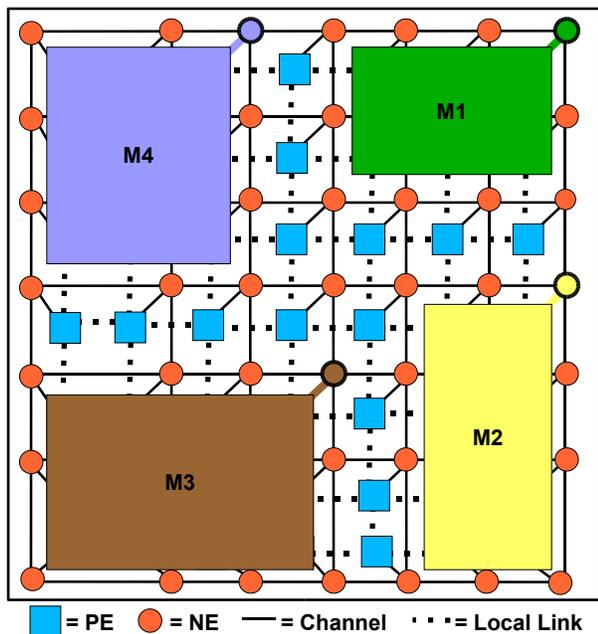
Figure 2. Temporal placement of four modules on the DyNoC

tion, the deactivated routers are reset to their default state.

3. S-XY-Routing

XY-routing is a deadlock free shortest path routing algorithm which routes packets first in X-dimension and then in the Y-dimension. In the DyNoC where the placement of modules alters parts of the mesh, thus producing "obstacles" for the routing, we have modified the classic XY-routing algorithm.

The new routing algorithm is called S-XY (Surrounding XY). Since it is an extension of the XY-routing algorithm it remains local-decisive[3] and deadlock free[4]. The routers operate in three different modes:

- The N-XY (Normal XY) mode. In this mode the router behaves as a normal XY router. A packet is first sent horizontally to the target column and then vertically to the target row.

- The SH-XY (Surround horizontal XY) mode. The router enters this mode, when its left neighbor or its right neighbor (horizontal neighbor) is deactivated.

- The SV-XY (Surround vertical XY) mode. The router enters this mode, when its upper neighbor or its lower neighbor (vertical) is deactivated.

3.1. Surrounding Obstacles in the X-direction

Assume without loss of generality, that a packet moving from right to left is blocked by an obstacle. There exist two alternative paths for the packet to reach its destination.

The first path is chosen if the Y-coordinate of the packet destination is greater or equal than that of the router address and the packet is sent upwards. Otherwise, the second path is chosen and the packet is sent downwards. One problem occurs when a packet with destination Y_{dest} is sent for example upwards and reaches a router node r with coordinate $Y_r > Y_{dest}$. According to the previous defined scheme the packet will be sent downwards to the router with coordinate $Y_r - 1$ which will send it upwards, thus producing a "ping-pong" game. To avoid this, the packet is stamped by setting a "stamp-bit" to notify router r not to send the packet back its incoming path. Upon reaching the router in the upper right of the obstacle, the stamp is removed and the packet is sent left, until its destination column or until another obstacle is found.

responding network graph to remain strongly connected [1] during the temporal placement. This condition is fulfilled if the following constraint is set when placing a given module: none of the surrounding routers must be covered by the module to be placed. Therefore, for each module placed onto the device, we will always find at least one line of network elements lying between any two modules. This is a sufficient condition for having a strongly connected network. Moreover, no packet will now be blocked in the network. As shown in Figure 2, each placed module is always surrounded by network elements, i.e. a ring of active network elements exists around each placed module at all times.

While in a static NoC [4, 9], each router always has four active neighbor router nodes[2], this is not always the case in the DyNoC presented for the first time in [5]. Whenever a module is placed on the device, it covers all routers in its area. Since these routers cannot be used, they will be deactivated until the module completes its execution by setting corresponding control signals. Upon completing its execu-

---

1   A network is strongly connected, if for each pair of network elements a path exists which connects these two elements.

2   The network elements at the device boundary are assumed to be connected to physical pins of the reconfigurable device.

---

3   The decision where to send a packet is taken at the local level.

4   Each packet will reach its destination after a finite number of steps.

3.2. Surrounding Obstacles in the Y-direction

The situation is different when a packet moving in the Y-direction is blocked. Assume without loss of generality that a packet moving from top to bottom is blocked by a placed module. Dealing with this case as with the previous one, the packet will be sent left or right. No preference is set here, because the packet is already in its right column. Let us assume that the packet is sent to the right, to the next router. Because the basic routing algorithm is the XY-routing, the next router will first compare the X position of the packet with its own position address. With the packet's X-destination being smaller, it will send the packet back to the router from which it received the packet. These two routers will keep sending the same packet to each other, thus creating a deadlock.

To avoid this "ping-pong" game again, we stamp the packet to notify all the routers above the obstacle that the packet is willing to surround the module. In our example the packet will then be sent right until the last router node above the module. There, the router removes the stamp and sends the packet downwards. From there on, we have the same situation as defined in the previous part (Surrounding Obstacles in the X-direction).

3.3. Deadlock free routing

In order to prove the S-XY algorithm to be deadlock free, we need to first prove that there is always a path from the packet source to the destination. Second, we must prove that each packet will reach its destination within a finite number of steps. The first requirement is guaranteed through the mentioned statement for strong connections among placed modules as described in Section 2.2. We now assume that a packet never reaches its destination. This will happen only if the packet is blocked or if the packet is looping in a given region. Because a path always exists from one active router to all other active routers, no packet can be blocked in the network, i.e. a packet is looping. Since this situation is not possible using the normal XY-routing algorithm, it can only arise while surrounding obstacles. But, when a packet is blocked in a given direction, it takes the perpendicular one. This is done until the last router on the module boundary which is at one corner of the module to be surrounded. From there, the normal XY routing resumes. The looping of a packet around a module is therefore not possible.

In the S-XY routing, fixing a priori for all routers the direction, where to send a packet whenever an obstacle is encountered can lead to extremely long routing paths like, caused by placements for which the routers always choose the extreme longest path.

To avoid this, each router is instructed by the placed module about the direction to take, whenever an incoming packet is blocked in a given direction by the module. Instead of using only one activation line, two lines are used in this case. The first line is used for the activation (1 = activate, 0 = deactivated) and the second one for the direction to take 0 = (east or south), 1 = (west or north). This considerably limits the complexity of the routers and there is no need for stamping anymore. We call this modification *router guidance* because the routers are guided by the modules.

4. Q-Routing

Loading and unloading of modules creates changing conditions and communication patterns. To optimize the performance of the dynamic network on chip, we need to adapt existing routing algorithms to support this dynamic environment.

Q-routing [6] is an adaptive packet routing algorithm for static networks, which is adapted here to dynamic networks on chip. The algorithm allows a network to continuously adapt to changing topology or congestion by routing packets on routes which estimate the least delivery time. During the entire operation of the network, data packets are sent only between directly adjacent routers. When a route becomes unavailable due to failure, router removal or congestion, Q-routing learns to avoid this route and chooses an alternate path. Therefore, we expect that Q-routing will work well in the context of the DyNoCs, where newly placed modules change the network topology by removing active network routers.

Q-routing is a distributed routing algorithm which learns a routing policy to minimize the delivery time of a packet to its destination. It performs its minimization by experimenting with different routing policies and gathering statistics about decisions which lead to minimal delivery time. Each router node in the network runs its own copy of the Q-routing algorithm using only local information from its direct neighbors.

When a router node x has to send a packet to a node d, it sends the packet to the neighbor y with the lowest estimated delivery time $Q^x(d, y)$. Using Q-routing, a router learns the expected delivery times to d for each adjacent router y, where an adjacent router is a direct neighbor of node connected to x by a network link. Each router node x keeps a two dimensional table Q^x of estimated delivery times which is updated after sending a packet according to the following formula:

$$Q_t^x(d, y) = (1 - \alpha)Q_{t-1}^x(d, y) + \alpha(b_t^x + \min_z Q_{t-1}^x(d, z))$$

The table Q^x has $n * m$ entries, where n is the number of accessible destination routers and m is the number of adjacent routers. The parameter α controls the learning rate and b_t^x is the time the current packet spent in x's buffer or queue before being sent at time step t. Immediately after sending the packet to neighbor y, router node x receives y's time estimate $min_{z \in neighbors\ of\ y}Q^x(d, z)$ to destination d. Once the router has learned the values associated with each destination-neighbor $(d,\ y)$, it executes a greedy policy. When a router node has to send a packet for node d, it sends this packet to the neighbor y with the lowest estimated delivery time $Q_t^x(d, y)$. Boyan and Littman have shown in [6] that a fast learning rate of $\alpha = 0.5$ worked well since it allows the network to adapt quickly.

Adapting Q-routing to the dynamic network on chip domain is straight forward. Here, the placement of a new module can cover up router nodes which have to be taken out of the network. Moreover, these routers are reactivated in the network after this module completes its execution. To address this, the delivery time to d via router y is set to ∞, i.e. $Q^x(d, y) = \infty$. When a router node reappears in the network, we optimistically set the estimated delivery time to d to 0, i.e. $Q^x(d, y) = 0$. This optimistic bias encourages exploration of new paths, as node x will always try to send packets via a node y which has just joined the network. If the result of this exploration does not yield a smaller delivery time, then node x will revert to its original behavior since $Q^x(d, y)$ will be updated to its true value. However, if this new router y can deliver packets in shorter time to their destination, then node x will continue to use y as its outgoing neighbor.

5. Evaluation

In the following case study, we evaluate the performance of the adapted XY-routing, as well as Q-routing on a 7×7 DyNoC array.

5.1. Simulator for DyNoCs

In order to evaluate the dynamic behavior of DyNoCs, we have implemented a simulator. SimDyNoC is a flexible and object oriented implementation of a discrete event network simulator written in `python`. Its main advantage is its ability to allow the removal of existing routers and its adjacent links during the run-time of the simulation. Due to the dynamic instantiation of routers and links, it is topology independent. This means that the number of links connected to each router node is not limited. Moreover, a link can connect any two routers inside the network if required.

Action	Description
R	Add a router to the network.
rR	Remove a router from the network.
rRC	Remove a router and all adjacent channels.
C	Add a channel between two routers.
rC	Remove a channel from network.
M	Place a module on to the network. Covered routers and channels are removed.
rM	Remove an existing module from the network and restore routers and channels which were covered by this module.
traffic	Specify the traffic pattern of one module to other placed modules.

Table 1. Supported actions in the simulation testbench

At each period of time, the simulator state reflects the state of a reconfigurable device on which modules are placed and removed dynamically.

The simulation configuration resides in the simulation testbench which describes all the time step-action pairs till the last time step. At each time step, modules can be placed or removed from the DyNoC. However, during the first time step, the basic DyNoC grid has to be created. This is done by instantiating routers and channels. A channel consists of two links which connect two routers together. Each input channel is connected to a FIFO where all incoming packets are stored. The size of the FIFO can be set for each link individually. Finally, application modules can be placed together with their respective traffic patterns at any desired time step. The list of possible actions is depicted in Table 1. It gives an overview over the most important actions supported by the simulation testbench. The most used action is traffic as it allows to dynamically adjust the communication patterns used for traffic generation between modules.

The router's arbiter is a central resource which resolves conflicts when two links compete for the same outgoing link. In this case, packets obtain control of the outgoing link based on round robin distribution. There is a one cycle routing delay per packet.

5.2. Experimental Results

The S-XY router implementation is straight forward in software as in hardware. The Q-routing as well as the S-XY-routing are implemented in our simulator. We implemented this S-XY-routing strategy in order to compare the performance and behavior with the Q-routing algorithm.

	4000 cycles	20000 cycles	50000 cycles
Sent Packets	11741	96759	261627
Throughput avg./max.	0.08 / 0.63	0.14 / 0.26	0.14 / 0.27
Network Load avg./max.	0.10 / 0.41	0.14 / 0.29	0.15 / 0.30
Latency avg./max.	4.82 / 14.65	4.80 / 13.00	4.82 / 13.00

Table 2. S-XY-routing performance values from simulation

	4000 cycles	20000 cycles	50000 cycles
Sent Packets	9500	96849	261714
Throughput avg./max.	0.15 / 0.51	0.15 / 0.43	0.15 / 0.43
Network Load avg./max.	0.23 / 0.49	0.16 / 0.58	0.16 / 0.58
Latency avg./max.	16.98 / 338	6.97 / 338.00	5.75 / 338

Table 3. Q-routing performance values from simulation

Both routing algorithms are compared on a 2D-mesh network, like the 7×7 configuration shown in Figure 1.
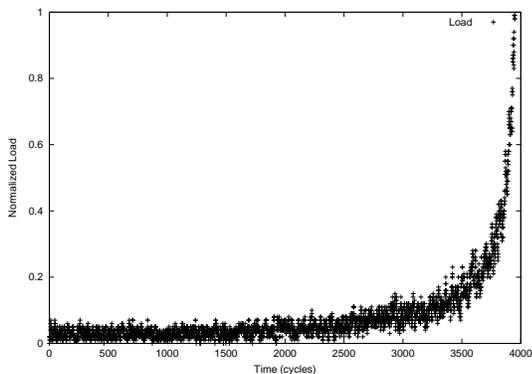

Figure 3. S-XY-routing load on a 7x7 DyNoC with active modules connected to each router

In the first evaluation, we use an increasing load function with uniform random communication pattern. This pattern is interesting as it varies the load on each router constantly while varying the packet destinations at the same time. Moreover, we must limit the load in the beginning of the simulation in order not to block the whole Q-routing network during its exploration phase. In the beginning, the low packet generation rate is held till the simulation cycle 1000. Thereafter, the probability for packet generation increases for every 20 simulation cycles. During each simulation cycle, a random router node is selected with the probability $P_a(t)$. When selected, this node generates a data packet with a random destination address. This packet is stored in a separate FIFO and can be sent in the next or following cycles, depending on the scheduler's decision. When this FIFO is full the generated data packet is discarded.

$$P_a(t) = \begin{cases} \frac{1}{155} & \text{if } 0 \leq t < 1000 \\ \frac{1}{155-(t-100)\div 20} & \text{if } t \geq 1000 \end{cases}$$

In our setup, network load is defined as the number of data packets held by the FIFOs in all network elements. Throughput is the number of transported data packets in the network. Network load, as well as throughput are normalized. Normalized load is defined to be the percentage of all queued packets divided by the sum of existing queue elements in the network. Similarly, normalized throughput is defined as the percentage of transported packets divided by the number of existing links.
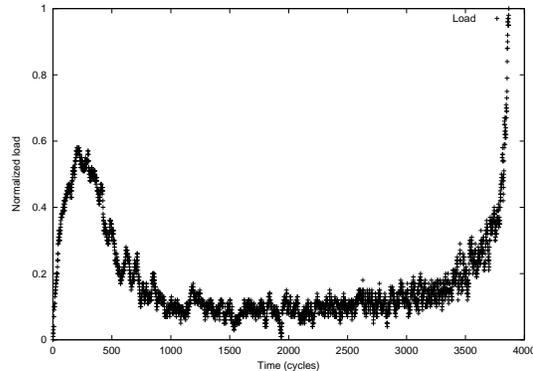

Figure 4. Q-routing load on a 7x7 DyNoC with active modules connected to each router

Figure 5 shows the latency during the S-XY-routing during the first 4000 cycles. The latency goes up to 14.65 cycles and achieves an average of 4.82 cycles during this simulation as shown in Figure 5. The load is under 0.10 in the first half of the simulation, as can be seen in Figure 3 and Table 2. All the simulations were performed with a 4 element deep FIFO buffer at each channel input. At around 3800 cycles,
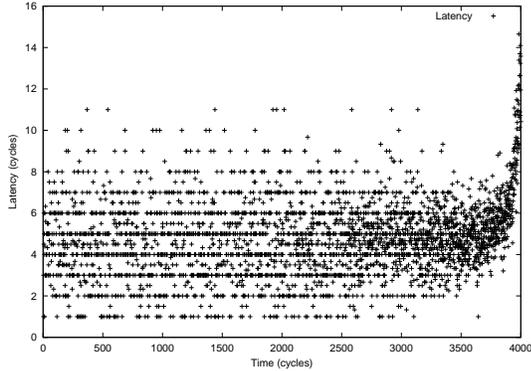
Figure 5. XY-routing latency on a 7x7 DyNoC with active modules connected to each router
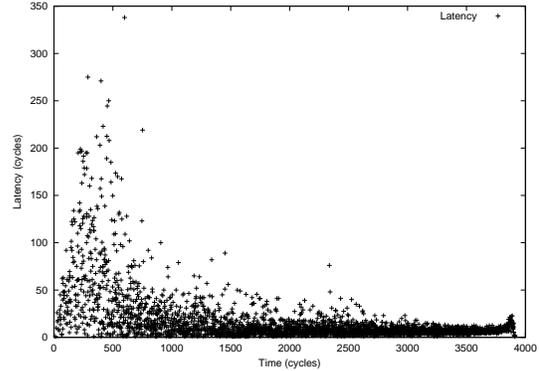

Figure 6. Q-routing latency on a 7x7 DyNoC with active modules connected to each router

the number of injected packets into the network reaches a critical point. Now both the load and latency increase without bound as the network enters the saturation point. During these 4000 cycles, the network manages to send and to deliver 11741 packets to their destination.

Q-routing behaves differently during the setup phase as it has to learn the shortest paths through exploration of the network. During the setup phase of the NoC, the data packets are traveling on sub-optimal paths (i.e. in general also longer paths with more hops) through the network, thus generating higher load in the individual routers and increasing the packet latency. This behavior is shown in Figure 6 and Figure 4. However, after a transient period of around 1000 simulation cycles the load settles down and remains at a constant level. From cycles 2500 on, the load curve of Q-routing is almost identical with the S-XY-routing curve from Figure 3. This similarity does not apply to the latency behavior. Here, approximately around cycle 3800, Q-routing saturates very quickly the network, whereas the S-XY-routing is just entering the saturation level. This shows that congestion avoidance and recovery [11] is required to prevent saturation from blocking the whole network.

To verify the performance of Q-routing, we have simulated these configurations for 20000 and 50000 cycles. We use the same increasing packet generation rate as in the 4000 cycle simulation, but we freeze the packet generation rate at time step 3500. The network load at this time step is shown in Table 4. This packet generation rate stays fixed till the end of the simulation. The following formula $P_b(t)$ defines the probability function used for packet generation in the second evaluation:

	Throughput	Network Load	Average Latency
S-XY-routing	0.20	0.20	3.20
Q-routing	0.19	0.19	3.75

Table 4. XY-routing and Q-routing performance values at time step 3500

$$P_b(t) = \begin{cases} \frac{1}{155} & \text{if } 0 \leq t < 1000 \\ \frac{1}{155-(t-100)\div 20} & \text{if } 1000 \leq t < 3500 \\ \frac{1}{30} & \text{if } t \geq 3500 \end{cases}$$

Comparing Table 2 and Table 3, we know that for this load level, Q-routing is able to send and deliver as many packets as S-XY-routing.

6. Summary and Future Work

This paper presents a concept for handling the problem of dynamic communication among modules which are dynamically placed on a reconfigurable device. Our contributions are the S-XY-routing and Q-routing algorithms which we have adapted for the dynamic environment with obstacles which are caused by dynamically placed modules.

Using simulation, we show that the S-XY, as well as the Q-routing algorithms are suitable for a dynamic network on chip implementation. Both algorithms have the advantage to depend on local information only. While the S-XY is simple and small, the Q-routing presents the learning advantage which quickly brings the system in a state of conver-

gence. The constant adjustment of the Q-routing algorithm enables the by-passing of local hot spots which is not possible with the S-XY-routing because all packets take a predefined path to destination. The drawbacks of the Q-routing are its setup phase and area costs. For a Q-routing implementation, the area cost is considerably higher, as Q-routing has to store the estimated delivery times in a table. However, the table size does not grow quadratically, as it is the product of number of placed modules, n and the number of adjacent router nodes m. With only ten placed modules and four adjacent router nodes, the table has forty elements. The high packet latency caused by the setup phase can be fully alleviated by preloading all router node tables with precomputed values when a predefined set of modules is loaded.

Still, there are important questions open: 1) Must the removal of routers caused through the placement of new modules delete the stored packets inside the FIFOs? 2) How must the hardware platform be constructed to efficiently support partial reconfiguration, module preemption and relocation? Moreover, future DyNoC hardware infrastructure concepts might be able to exploit the interplay between routing aware module placement and load aware routing algorithms in order to minimize the overall load in the network.

7. Acknowledgments

References

[1] A. Ahmadinia, C. Bobda, and J. Teich. A dynamic scheduling and placement algorithm for reconfigurable hardware. In C. Müller-Schloer, T. Ungerer, and B. Bauer, editors, *Proc. of 17th International Conference on Architecture of Computing Systems (ARCS 2004)*, volume 2981 of *Lecture Notes in Computer Science (LNCS)*, pages 125–139, Augsburg, Germany, Mar. 2004. Springer.

[2] K. Bazargan, R. Kastner, and M. Sarrafzadeh. Fast template placement for reconfigurable computing systems. *IEEE Design and Test - Special Issue on Reconfigurable Computing*, January-March:68–83, 2000.

[3] L. Benini and G. Micheli. Network on chips: A new soc paradigm. *IEEE Computer*, Jan. 2001.

[4] D. Bertozzi and L. Benini. Xpipes: A network-on-chip architecture for gigascale system-on-chip. *IEEE Circuits and Systems Magazine*, Second Quarter:18–31, 2004.

[5] C. Bobda, M. Majer, D. Koch, A. Ahmadinia, and J. Teich. A dynamic NoC approach for communication in reconfigurable devices. In *Proceedings of International Conference on Field-Programmable Logic and Applications (FPL)*, volume 3203 of *Lecture Notes in Computer Science (LNCS)*, pages 1032–1036, Antwerp, Belgium, Aug. 2004. Springer.

[6] J. A. Boyan and M. L. Littman. Packet routing in dynamically changing networks: A reinforcement learning approach. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 671–678. Morgan Kaufmann Publishers, Inc., 1994.

[7] W. J. Dally and B. Towles. Route packets, not wires: On-chip interconnection networks. In *Proceedings of the Design Automation Conference*, pages 684–689, Las Vegas, NV, June 2001.

[8] A. Hemani, A. Jantsch, S. Kumar, A. Postula, J. Oberg, M. Millberg, and D. Lindqvist. Network on chip: An architecture for billion transistor era. In *Proceeding of the IEEE NorChip Conference*, Nov. 2000.

[9] T. Marescaux, J.-Y. Mignolet, A. Bartic, W. Moffat, D. Verkest, S. Vernalde, and R. Lauwereins. Networks on chip as hardware components of an OS for reconfigurable systems. In *Proceedings of 13th International Conference on Field Programmable Logic and Applications*, Lisbon, Portugal, Sept. 2003.

[10] C. Steiger, H. Walder, M. Platzner, and L. Thiele. Online Scheduling and Placement of Real-time Tasks to Partially Reconfigurable Devices. In *Proceedings of the 24th International Real-Time Systems Symposium*, Dec. 2003.

[11] M. Thottethodi, A. Lebeck, and S. Mukherjee. Exploiting global knowledge to achieve self-tuned congestion control for k-ary n-cube networks. In *IEEE Transactions on Parallel and Distributed Systems*, volume 15, pages 257 – 272, Mar. 2004.

[12] H. Walder, C. Steiger, and M. Platzner. Fast online task placement on FPGAs: Free space partitioning and 2d-hashing. In *Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS) / Reconfigurable Architectures Workshop (RAW)*, page 178. IEEE Computer Society, Apr 2003.