APPLYING PARTIAL RECONFIGURATION TO NETWORKS-ON-CHIPS

Thilo Pionteck, Roman Koch, and Carsten Albrecht

University of Lübeck
Institute of Computer Engineering
23538 Lübeck, Germany
email:{pionteck, koch, albrecht}@iti.uni-luebeck.de

ABSTRACT

This paper presents CoNoChi, an adaptable Network-on-Chip for dynamically reconfigurable hardware designs. CoNoChi is designed for taking advantage of the partial dynamic reconfiguration capabilities of modern FPGAs and applies this feature to adapt the network structure to the location, number and size of currently configured hardware modules. The network consists of the minimal number of switches required. Switches can be added or removed from the network by a global control instance at runtime. Compared to common fixed Network-on-Chip structures, the CoNoChi architecture reduces the area requirements and latency of the network and eases the online placement of hardware modules. Two variants of CoNoChi are presented: one is based on a homogeneous hardware structure that is dynamically reconfigurable on logic block level, and the other one is adapted to the limited partial reconfiguration capabilities of Xilinx Virtex-II (Pro) FPGAs.

1. INTRODUCTION

In combination with Intellectual Property (IP) based System-on-Chip (SoC) design, Networks-on-Chips (NoCs) are often proposed as a promising communication architecture for addressing the challenges of complex system designs. NoCs provide a better bandwidth utilization than bus based architectures and the modular structure of NoCs eases the integration of different IP-cores [1]. At design time, the scalability of NoCs is used to adapt the network to the performance and area constraints of the design [2, 3]. If these constraints are fixed, as it is the case for ASIC or classical FPGA designs, an optimized communication network for one particular SoC can be designed. With the emergence of dynamically reconfigurable FPGAs, system constraints are not necessarily fixed anymore. Hardware modules can be exchanged at runtime, leading to changing communication requirements. A fixed NoC has to be designed that complies with the communication requirements of all possible combinations of hardware modules. If the number, size and location of hardware modules can change, too, a huge amount of communication resources has to be provided to cover all possible configurations.

CoNoChi (Configurable Network-on-Chip) addresses this problem by using the dynamic partial reconfiguration capabilities of modern FPGAs to adapt the structure of the NoC to the number and location of currently configured hardware modules. By only providing the minimal number of switches required, both latency and area requirements of the network are minimized. The flexible communication network facilitates an arbitrary placement of hardware modules of rectangular shape and arbitrary size, easing a high device utilization. For the reconfiguration process, CoNoChi uses the same mechanisms as required for the dynamic exchange of hardware modules. Thus, the additional hardware overhead introduced by the reconfiguration capabilities of CoNoChi is comparatively low. First ideas of CoNoChi were presented in [4, 5]. For this contribution, the focus is set on the hardware realization of CoNoChi based on a prototypical implementation on a Xilinx Virtex-II Pro. The routing strategy is discussed as well as the special requirements of the dynamic network structure to the protocol level of the network.

The outline of the paper is as follows: Section 2 discusses related work on communication structures for reconfigurable designs. In Section 3, the concept of CoNoChi is presented. Its hardware structure on a homogeneous and fine-grained reconfigurable hardware platform is given in Section 4. Section 5 provides information of the adaption of CoNoChi to the limited partial reconfiguration capabilities of a Xilinx Virtex-II Pro FPGA and discusses prototyping results. Concluding remarks are given in Section 6.

2. RELATED WORK

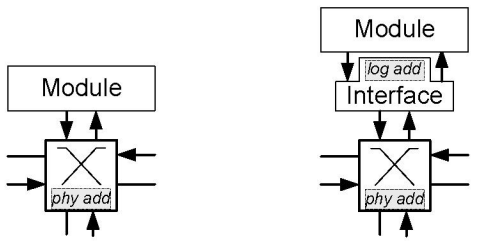
Although many works exist on *configurable* NoCs, most of them refer configurability to the design time of the system, not as configurability at runtime. Some *configurable* NoCs allow dynamic adaptation of routing strategy or quality of service, but only few projects deal with dynamic communication structures for reconfigurable modules. The majority

1-4244-0 312-X/06/$20.00 ©2006 IEEE.

of works concerning this problem use either static NoCs or bus based communication architectures like [6, 7], but only a limited number of NoC architectures exist which are able to change their topology at runtime. One of them, DyNoC (Dynamic Network on Chip) was introduced by Bobda et al. [8]. The DyNoC architecture consists of an array of processing elements surrounded by a huge number of switches. At runtime, switches can be disabled and their hardware resources can be reused for dynamically inserted hardware modules. Routing only depends on the position of the destination and is realized using an extension of the *XY* algorithm which is capable of surrounding obstacles. This approach mitigates the problems of fixed locations and fixed sizes of exchanged hardware modules and reduces the area requirements of the network compared to static NoCs. The CoNoChi architecture proposed in this paper consists of the minimal numbers of switches required. When hardware modules are inserted or removed by partial dynamic reconfiguration the number of switches and their location are adapted. This leads to further area savings compared to DyNoC and reduces the latency of the network by minimizing the number of hops. Other differences are the multi-layer protocol stack and the routing mechanism which are explained in the following.

3. CONOCHI: CONFIGURABLE NETWORK ON CHIP

The NoC consists of virtual cut-through switches with four equal full-duplex links. The links can be either used for connections to adjacent switches or for the connection of one hardware module to a switch. Depending on the number of non-local links a linear network or a two-dimensional grid can be instantiated. CoNoChi supports packets of up to 1036 bytes with a packet header of 12 bytes. The packet header is divided into three layers: data-link layer, network-layer and application layer. Inside the switches only the data-link layer is processed. For a detailed description of the header please refer to [9].



(a) Direct connection (b) Connection via an interface

Fig. 1. Variants of connecting hardware modules to switches

During design time, CoNoChi can be configured in different ways. One parameter is the bit width of all links which can be either 8, 16 or 32 bit, another parameter is the way how hardware modules are connected to a switch. There are two variants. As shown in figure 1(a) variant A directly connects hardware modules to switches whereas variant B (see figure 1(b)) uses a special interface to connect to the hardware module. While variant A is commonly used for NoCs, variant B enables some special features of CoNoChi: support of physical and logical addresses, packet forwarding and support of several tasks within one hardware module.

3.1. Physical and logical addresses

In general, NoCs only use a physical destination address for routing. In CoNoChi this address is provided in the data-link layer of the packet header. In addition to this, the network layer of the packet header also provides logical addresses which are evaluated by the interface module for connecting a hardware module to a switch. Physical addresses refer to specific switches at specific locations within the NoC topology. Logical addresses refer to processing entities inside hardware modules. Packets are routed from one switch to another solely based on the physical address using dynamic routing tables. If the physical address of a switch matches, the packet is passed on to the interface module. The interface module, in turn, determines by the logical address in the packet header which of its processing entities is addressed. A single hardware module and thus its assigned interface may listen to multiple logical addresses. The differentiation between physical and logical addresses allows hardware modules and sub-ordinate processing entities to be moved or combined within the NoC. Moreover, the protocol is prepared for an envisioned multi-chip system which comprises one NoC per chip. Here, physical addresses would be local to each chip and refer to switches as well as to inter-chip gateways while logical addresses remain valid in the system level context.

3.2. Message classes

Another feature of CoNoChi is its support of quality of service and its support of several message classes. For distinction, a *type* and a *priority* field in the data-link layer of the packet header are used. An overview of the different message classes is given in table 1. The *payload* class is used to transport user data to and from hardware modules. If state information of hardware modules is required by other system components the *monitor* class is used. An example would be a reconfiguration manager, which collects workload information of hardware modules in order to determine whether copies of already instantiated hardware modules need to be configured. The *Log_Addr* class provides interface modules with their logical addresses as well as with physical and logical addresses for packet forwarding. Packet forwarding is required when addressed process-

Type	Prio	Class	Description
00	00	Payload	Payload for hardware modules
00	01	Monitor	State inform. of hardware modules
01	10	Log_Addr	Control data for interface
10	10	Route_Tbl	Routing tables for switches
10	11	NoC_Msg	Internal message for flow control
11	11	Rec_Data	Data for dynamic reconfiguration

Table 1. CoNoChi message classes [5]

ing entities have moved to other physical locations (see Section 3.4) and for forwarding processed packets to their new destination. In the latter case the *trans* field inside the application layer of the packet header is used to select one of the physical/logical address pairs stored inside the interface unit. Routing tables are transmitted using the *Route_Tbl* class. Routing information consists of two bits per physical address and can be transmitted individually so that only updates have to be provided. The *NoC_Message* class is used for NoC internal messages, e.g. to enable/disable links of a switch. Reconfiguration data can also be transmitted via the NoC, using the *Rec_Data* class. This class is mainly provided for an envisioned multi-chip NoC.

3.3. Routing

CoNoChi applies an oblivious, deterministic routing scheme for a certain network topology. The routing decisions are solely based on off-line topology information. Routing tables are generated at a central instance and distributed via the network along a tree-based overlay network. This overlay network corresponds to a minimal-spanning tree with the central instance as the root. New calculations of routing tables are triggered by topology changes caused by dynamic reconfiguration. The metric considered for routing applies fixed and equal costs per link. The goal for all routes are minimal delays caused by a minimum number of hops. Thus, the routing tables are created by solving the All-Pairs Shortest-Path Problem for the graph representing the current network topology. The Floyd-Warshall algorithm [10] is used here although its complexity $O(n^3)$ (with n nodes) is rather high. Nevertheless, this algorithm suits fine because of its compact coding capability and a relatively small number of nodes in a typical NoC. In an envisioned multi-chip NoC the increasing runtime of the algorithm can be reduced by dividing the whole network into subnetworks connected by border routers. Subnetworks are treated on their own.

Concerning deadlock and livelock freedom the initial routing strategy had to be extended. Although shortest-path algorithms normally provide livelock freedom, livelocks may occur as the topology of CoNoChi may change during runtime. By ensuring that the time between two reconfigurations is longer than the maximum lifetime of network packets, livelocks can easily be avoided. The problem of deadlocks is specific to shortest path algorithms. Deadlocks can occur if the network topology contains cycles. In CoNoChi, this problem is addressed by using a method based on ordered-trees as presented in [11].

3.4. Runtime Reconfiguration

CoNoChi provides a dynamic communication infrastructure for reconfigurable modules. This comprises support of dynamic replacement of hardware modules as well as dynamic insertion and removal of switches. Dynamic replacement of hardware modules can be done in two different ways: with and without stalling the NoC. For the first variant, no special precautions have to be taken. As the NoC is stalled, no packets are circulating in the network at the time of reconfiguration. Consequently, no packets can be delivered to a wrong hardware module after dynamic reconfiguration although the physical address of the switch connecting the old and the new hardware module did not change. This assumption cannot be made in the second case where the NoC is not stalled. Here, packets that are destined for the original hardware module may still be in transit. These packets have either to be deleted or assigned to another processing unit which can provide the same functionality as original hardware module, e.g. software based hardware modules. In both cases the interfaces of exchanged hardware modules have to be informed how to proceed with packets having an invalid logical address. Therefore, packet forwarding as stated earlier in this section is required.

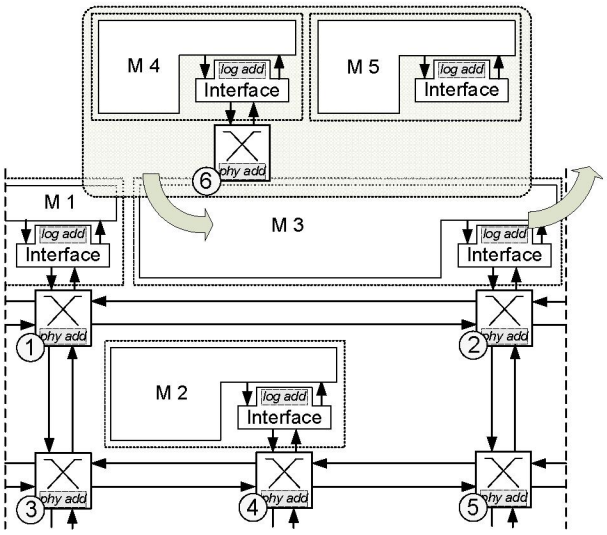


Fig. 2. Dynamic insertion of hardware modules (adapted from [4])

Dynamic insertion and removal of switches is more complex. A typical scenario is depicted in figure 2, were

hardware module 3 (M3) plus its interface is replaced by two smaller hardware modules (M4, M5) and their interfaces. As switch ② can only be used for connecting one of the interfaces of the new hardware modules, a new switch ⑥ has to be provided for the connection of the interface of M4 to the network. This switch is placed in the link between switches ① and ②. During this process, packets normally sent between switch ① and ② have to be redirected via switch ③, ④, and ⑤. This bypass is set up by updating the routing tables of the corresponding switches, which is done by the global control instance of the system. With these updates of the routing tables, switch ① also obtains knowledge of the new switch ⑥ and its physical address, thus enabling access to switch ⑥ after reconfiguration. The last step before starting the reconfiguration process is to disable the respective links of switches ① and ② as invalid signals may occur during partial dynamic reconfiguration. After partial reconfiguration, switch ⑥ has to be initialised and the link between switches ①, ⑥ and ② has to be (re)activated. Therefore, new routing tables are sent by the global control instance to the corresponding switches. Note that the feature of dynamic reconfiguration without stalling the NoC causes some restrictions to the topology of the network.

4. HARDWARE ARCHITECTURE

Realization of CoNoChi depends very strongly on the characteristics of the underlying hardware platform. To make use of all features CoNoChi offers, a homogeneous FPGA that is dynamically reconfigurable on logic block level is required. Homogeneity is needed to move hardware blocks arbitrarily from one location to another without modifying the configuration data. Therefore routing resources as well as the structure of the logic cells need to be homogeneous. Special components like memory blocks or hierarchical routing resources can be included, but their occurrence needs to be regular so that closely meshed grids can be formed. The capability of partial reconfiguration on logic block level is needed to manipulate arbitrary rectangular subareas of the FPGA during runtime while hardware modules located in the other areas of the FPGA are still active. In particular, partial reconfiguration should not affect communication between modules which are not reconfigured. Unfortunately, these requirements are only partially fulfilled in actual FPGAs like Xilinx Virtex-II (Pro). For the design of CoNoChi we oriented on the capabilities of Xilinx Virtex-II Pro FPGAs as far as possible, but especially concerning the partial reconfiguration capabilities a more flexible architecture was assumed. These idealizations are similar to the ones made in works on online placement of modules in a 2D reconfigurable area like in [12] and seem to be realistic for future devices. A way to overcome some limitations of actual FPGAs for a realization of CoNoChi is given in section 5.

The main hardware component of CoNoChi is the switch. The global control instance of CoNoChi, which is responsible for providing the routing tables, dispatching of incoming packets, and the analysis of state information of hardware module (if applicable, see section 3.2), is realized in software. Using a Xilinx Virtex-II Pro as hardware platform, its embedded PowerPC core can be used for these tasks. A third component of CoNoChi is the optional interface for connecting hardware modules to switches. As the structure of the interface depends on the individual hardware module and application area of the system, its structure is not discussed in this contribution. Readers may refer to [5].

For each of the four input ports of a switch a FIFO capable of storing a complete packet is used. For a Virtex-II Pro realization, Block RAMs are utilized as memory elements. When a new packet is received, at first its priority has to be determined. Therefore the *prio* and the *type* field of the datalink layer of the packet header are bypassed to the arbiter which regulates the access to the routing table and to the controller in the case of NoC internal messages. In the case of equal priorities, access is granted round robin. The routing table consists of 64 entries of 2 bit and can be mapped onto Distributed RAM in a Virtex-II FPGA. Depending on the physical destination address a port is assigned and the packet is immediately forwarded. In total five clock cycles are required for packet processing inside a switch.

Beside the physical address all switches are identical. Thus, only one configuration code for all switches is required. If a switch is added to the network at runtime, the frame position and the part of the configuration code containing the physical address have to be updated by the global control instance. After dynamic insertion, the new switch needs to be initialized. The reset signal has to be applied exclusively to the new switch so that other system components are not affected. This can be done by either routing a local reset signal to the switch or by making use of an actually unwanted feature of the Virtex-II FPGA family, the reinitialization of Distributed RAM after a configuration. Routing a local reset signal to the switch is only feasible if the signal comes from a direct neighbour switch. In this case the neighbour switch has to be informed by a NoC internal message to activate/deactivate the corresponding signal. If the local reset signal would be applied by the global control instance, a new signal had to be routed from the global control instance to the switch, affecting also non changing parts of the system. Local reset lines can be avoided by making use of the fact that in Virtex-II FPGAs the content of Distributed RAM is reinitialized by a full bitstream [13]. The output of a Distributed RAM element is used to drive a switch internal reset signal which causes an FSM to invert the corresponding memory element. Thus a local reset signal is automatically generated without need for an additional reset signal.

5. PROTOTYPING

A prototype of CoNoChi was realized using a Xilinx Virtex-II Pro 100 FPGA. Due to the limited reconfiguration capabilities of the device, some restrictions concerning the functionality of CoNoChi had to be made. The main drawback of the Virtex-II architecture is the fact that only complete columns of the device can be reconfigured at runtime. Thus, no arbitrary 2D positioning of switches or hardware modules can be done. However, the Virtex-II architecture offers the capability of glitchless reconfiguration [14]. Except for Distributed RAM and SLR16 components, dynamic reconfiguration does not affect functionality of components if their configuration data is kept. Hence, by only applying partial changes of the configuration data of a column, a limited 2D reconfiguration capability can be emulated. Another characteristic of the Virtex-II architecture also hampers an arbitrary 2D positioning of switches or hardware modules. The hierarchical routing resources of CLBs differ although their appearance is periodical. The same applies to the Block RAM. As one switch consists of four Block RAMs and about 100 CLBs, different configuration codes for each combination of the two patterns would be required. To reduce the amount of configuration data and to ease the reconfiguration process, the positioning of switches and modules is limited to a fixed grid. This leads to a structure as shown in figure 3.

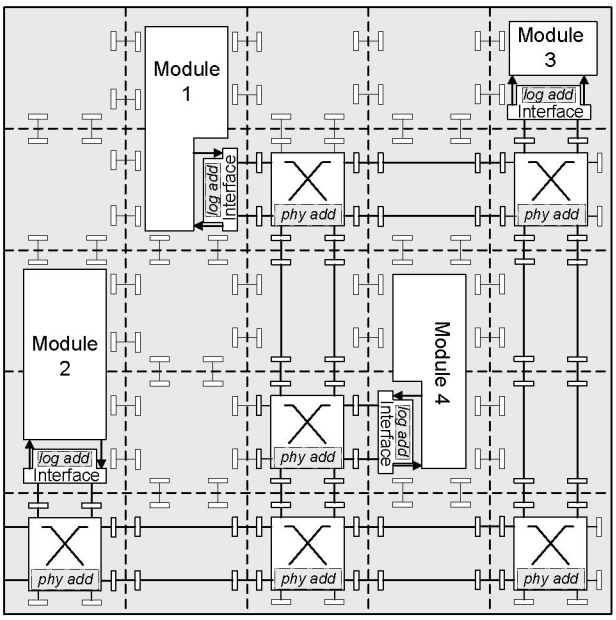


Fig. 3. CoNoChi structure on Virtex-II FPGA

The FPGA is partitioned into a grid of rectangular subareas. Each subarea can be configured as a switch, as a horizontal or vertical communication line or as part of a hardware module. The configuration data for the different variants are stored in an external memory. In order to avoid online routing, each subarea is configured to contain default communication lines crossing the border of adjacent subareas, with starting points and endpoints routed to LUTs. Dynamically exchanged hardware modules, switches or communication lines can connect to these LUTs and thereby get access to the NoC. This mechanism implies that all unused subareas are configured to contain at least these default communication lines and LUTs.

With these work-arounds CoNoChi can be prototyped on a Virtex-II (Pro) FPGA. One problem, however, still exists. Switches cannot access their routing table while a partial reconfiguration affecting one of their CLB columns is in progress. Although the capability of glitchless reconfiguration guarantees a continuous operation of unchanged CLBs, a conflict emerges as routing tables of switches are stored in Distributed RAM. These memory components cannot be used while a reconfiguration is in progress. So no incoming packets can be routed. The only way to overcome this problem is to provide a cut-through mode for switches which directly connects horizontal links of switches. This mode can be activated by the global control instance using NoC internal messages.

	Optimized for			
	Speed		Area	
Bit width	Slices	Freq. $[MHz]$	Slices	Freq. $[MHz]$
8	463	90	363	72
16	479	111	397	66
32	493	88	410	73

Table 2. Synthesis Results of a switches for different bit widths and optimization criterias

Synthesis results for placed and routed switches of different bit widths and different optimization criterias are given in table 2. The values refer to an implementation on a Virtex-II Pro 100 with speed grade 6 and 44096 slices in total. In addition to the given numbers of slices each switch requires four Block RAMs. The critical path for all variants of the switch is inside the switch controller. By spending an additional clock cycle for the packet processing, the clock frequency can easily be increased. The actual latency of a switch is five clock cycles. A possible layout of CoNoChi for a bit width of 16 bit is shown in figure 4. For this example, the partial bitstream for inserting a switch comprises about 238 frames with a total size of about 297 Kbyte. Neglecting the time to load the configuration data from an external memory, dynamic inserting or removing of a switch requires about $6.1ms$. In the meantime all other preparations for updating the NoC can be done, e.g. computation of routing tables. Using the embedded PowerPC 405 core,

an implementation of the Floyd-Warshall algorithm used for routing requires about 1600 bytes instruction memory and 4 Kbytes data memory for computation and thus can be performed within the PowerPC caches. Its runtime can be neglected with regard to the time required for partial reconfiguration.

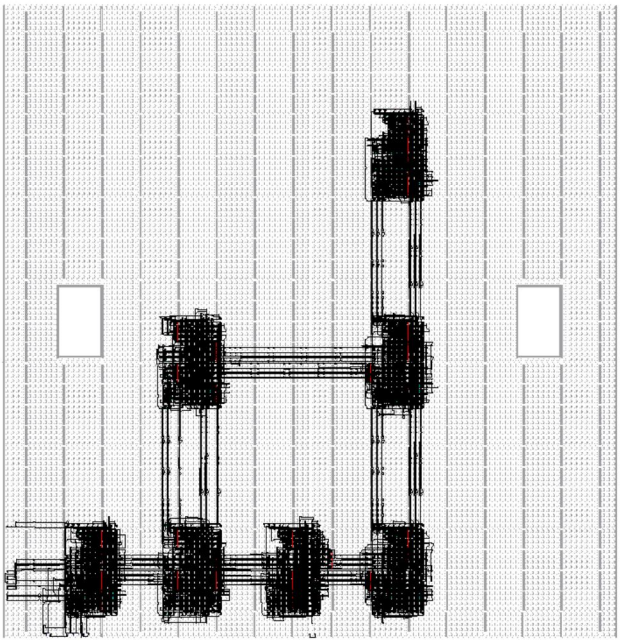


Fig. 4. Layout of CoNoChi on a Virtex-II Pro 100 FPGA

6. CONCLUSION

In this paper, a NoC architecture for dynamically reconfigurable hardware platforms was presented. The architecture enables the adaptation of the network topology at runtime and thus the customization of the communication resources to the currently configured hardware modules. The theoretical concept was demonstrated as well as a prototypically implementation on a Virtex-II Pro FPGA, though this device features 1D partial reconfiguration only. With the practical availability of the Virtex-4 FX FPGAs and their improved partial reconfiguration capabilities, limitations and work-arounds of the Virtex-II Pro implementation can be avoided.

7. ACKNOWLEDGEMENT

This work was funded in part by the German Research Foundation (DFG) within priority programme 1148 under grant reference Ma 1412/5.

8. REFERENCES

[1] L. Benini and D. Bertozzi, "Network-on-chip architectures and design methods," *IEE Proc. of Computers and Digital Techniques*, vol. 152, no. 2, pp. 261–272, Mar. 2005.

[2] R. Cardoso, M. Kreutz, L. Carro, and A. Susin, "Design space exploration on heterogeneous network-on-chip," in *IEEE Int. Symposium on Circuits and Systems*, vol. 1, May 2005, pp. 428–431.

[3] T. Hollstein, H. Zimmer, and M. Glesner, "Dynamic hardware/software co-design based on a communication-centric hyper-platform," in *Proc. ICM 6th Int. Conference on Microelectronics*, Dec. 2004, pp. 355–358.

[4] T. Pionteck, R. Koch, and C. Albrecht, "A Dynamically Reconfigurable Packet-Switched Network-on-Chip," in *DATE '06: Proc. of Design Automation and Test in Europe*, Mar. 2006, pp. 136–137.

[5] R. Koch, T. Pionteck, C. Albrecht, and E. Maehle, "An Adaptive System-on-Chip for Network Applications," in *Proc. of 13th Workshop on Reconfigurable Architectures*, Apr. 2006, p. 194.

[6] M. Hübner, T. Becker, and J. Becker, "Real-time LUT-based network topologies for dynamic and partial FPGA self-reconfiguration," in *Proc. 17th Symposium on Integrated Circuits and System Design*, Sept. 2004, pp. 28–32.

[7] A. Ahmadinia, C. Bobda, J. Ding, M. Majer, J. T. S. Fekete, and J. van der Veen, "A practical approach for circuit routing on dynamic reconfigurable devices," in *16th IEEE Int. Workshop on Rapid System Prototyping*, June 2005, pp. 84–90.

[8] C. Bobda and A. Ahmadinia, "Dynamic Interconnection of Reconfigurable Modules on Reconfigurable Devices," *IEEE Design & Test of Computers*, vol. 22, no. 5, pp. 443–451, Oct. 2005.

[9] C. Albrecht, R. Koch, and T. Pionteck, "On the Design of a Loosely-Coupled Run-Time Reconfigurable Network Coprocessor," in *Proc. of the Workshop on Media and Streaming Processors 2005 (MSP-7)*, Nov. 2005.

[10] R. W. Floyd, "Algorithm 97: Shortest path," *Communications of the ACM*, vol. 5, no. 6, p. 345, June 1962.

[11] D. M. Chiu, M. Kadansky, R. Perlman, J. Reynders, G. Steele, and M. Yuksel, "Deadlock-free routing based on ordered links," *Local Computer Networks*, pp. 62–71, Nov. 2002.

[12] H. Walder, C. Steiger, and M. Platzner, "Fast online task placement on FPGAs: free space partitioning and 2D-hashing," in *Proc. Inter. Parallel and Distributed Processing Symposium*, Apr. 2003.

[13] *Virtex-II Pro and Virtex-II Pro X FPGA User Guide*, Xilinx, Inc., Mar. 2005.

[14] B. Blodget, C. Bobda, M. Hübner, and A. Niyonkuru, "Partial and Dynamically Reconfiguration of Xilinx Virtex-II FPGAs," in *Proc. of 14th Int. Conference Field Programmable Logic and Application*, Aug. 2004, pp. 801–810.