ADAPTABLE SWITCH BOXES AS ON-CHIP ROUTING NODES FOR NETWORKS-ON-CHIP

Ralf Eickhoff, Jörg-Christian Niemann, Mario Porrman, Ulrich Rückert
Heinz Nixdorf Institute, System and Circuit Technology, University of Paderborn
Fürstenallee 11, 33102 Paderborn, Germany
{eickhoff,niemann,porrmann,rueckert}@hni.upb.de

Abstract Due to continuous advancements in modern technology processes which have resulted in integrated circuits with smaller feature sizes and higher complexity, current system-on-chip designs consist of many different components such as memories, interfaces and microprocessors. To handle this growing number of components, an efficient communication structure must be provided and incorporated during system design. This work deals with the implementation of an efficient communication structure for an on-chip multiprocessor design. The internal structure of one node is proposed and specified by its requirements. Furthermore, different routing strategies are implemented. Moreover, the communication structure is mapped on a standard cell process to examine the achieved processing speed and to determine the area requirements.

Keywords: Switch Box, Networks-on-Chip, SoC, Multiprocessor Architectures.

Introduction

Due to upcoming improvements in semiconductor processes it is possible to integrate a huge amount of components on a single chip. Consequently, different IP cores have to cooperate and to communicate with each other. Thus, when designing the system, an efficient communication structure must be provided, which is able to manage the traffic between all components. On the one hand, many existing approaches known from computer networks can be adapted to networks-on-chip (NoCs). For example, a circuit-switching network or a packet-switching network can be established and the performance and flexibility trade-off has to be solved. On the other hand, these networks-on-chip differ from traditional computer networks. In a packet-switched network-on-chip, for example, the packet size will mostly be much smaller than in a computer network. Furthermore, the geometrical dimensions of such a network are much smaller compared to computer networks. Consequently, lower latency and higher throughput can be achieved. In this paper we present a network-on-chip, which is based on the packet switching approach. It is able to handle the traffic between an unlimited but known number of modules.

After a short account on how the design of the network-on-chip was motivated, the implementation is presented in Section 1 where the topology of the network and the topology of each node are shown. In Section 2 two different routing strategies are analyzed. In Section 3 we present first results of a synthesis of this network structure.

Motivation

Due to increasing traffic in computer networks and to the growth of the whole network there is an increasing demand to manage the traffic. For this task, a general purpose processor is not efficient for the required performance whereas an application-specific instruction processor (ASIP) such as a network processor (NPU) is more suitable. Today, these complex architectures can be integrated into one single chip due to modern design techniques, as mentioned in the introduction.

In the GigaNetIC project [6] we aim at developing high-speed components for networking applications based on massively parallel architectures. A central part of this project is the design, evaluation, and realization of a parameterizable network processing unit. The proposed architecture is based on massively parallel processing, enabled by a multitude of processors, which form a homogeneous array of processing elements arranged in a hierarchical system topology with a powerful communication infrastructure. Four processing elements are connected via an on-chip bus to a so-called switch box, cf. fig. 1, which allows a forming of arbitrary on-chip topologies. Hardware accelerators support the processing elements to achieve a higher throughput and help to reduce energy consumption. Following a top-down approach, network applications are analyzed and partitioned into smaller tasks. The tasks are mapped to dedicated parts of the system, where a parallelizing compiler exploits inherent instruction level parallelism. The hardware has to be optimized for these programming models in several ways. Synchronization primitives for both programming hierarchies have to be provided and memory resources have to be managed carefully. Furthermore, the shown dimension of the system is only one example of our multiprocessor system. The number of parallel operating processors can be further increased. As a core component of our architecture, we use a 32 bit RISC CPU, the S-Core, which has been designed in our research group [5]. By changing the number of ports of the switch boxes the topology of the on-chip network can be arbitrarily formed including meshes, butterfly networks or tori [1]. For our first implementation we have chosen a mesh topology due to efficient hardware integration (cf. section 3).

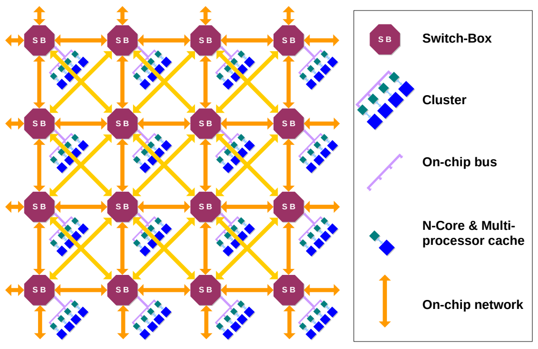


Figure 1. Architecture of the network processor proposed in the GigaNetIC project

1. Implementation of the network-on-chip

As in computer networks, different requirements have to be considered in order to find an efficient implementation for a special application. An important issue of a common network is the underlying graph. This has a strong impact on several parameters of the network affecting latency, throughput etc. When the topology of the network is fixed each switch box can be designed due to its requirements. The inputs for example, have to face tasks like storing and delivering packets. In the following these packets will be referred to as flits, which represent the atomic transportation units in our approach.

1.1 Topology of the network

The network-on-chip can be evaluated with the methods known in computer networks. Consequently, the existing models can be adapted to characterize the network and the topology can be described by a graph. Besides common similarities in respect to the modeling of a network, some differences are still present. Compared to a computer network, a network-on-chip is marked by several differences [4]:

- different geometrical dimension; a system-on-chip obtains short distances to other nodes
- parallel communication structures are easier to establish on a chip without high costs
- higher bandwidth due to parallel communication structures
- deterministic and periodic traffic
- smaller packet size

Consequently, these characteristics have to be considered in the network topology. For the application as a network processor the on-chip network has to meet the following requirements:

- regular communication structure
- high scalability during design time
- good re-use
- high performance

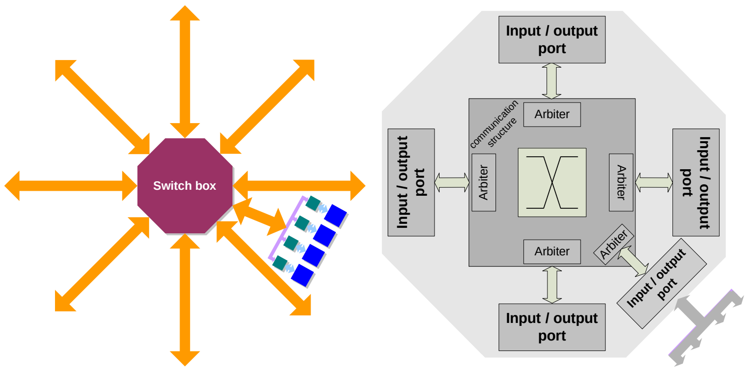
Moreover, as the production process maps the communication structure on a two-dimensional surface a regular communication structure is provided by a two-dimensional graph. Thus, the 2-d grid, the 2-d torus and the binary tree are well suited to achieve a regular communication structure. The binary tree provides a small bisection bandwidth which results in high throughput requirements next to the root of the tree. Thus, a communication link that allows higher throughput in comparison to the other links of the net has to be established. This also leads to an irregular structure.

The grid and the torus provide a regular communication structure and have no technical limits in respect to a growing number of nodes. Consequently, these structures can be used when a new design of the network processor contains more computing arrays and thus more nodes. The main disadvantage of a torus is the circular connection. This causes longer interconnection delays at the boundaries, and leads to lower operating speed in a globally synchronous design.

We have chosen the grid as the topology for the network-on-chip. In order to provide a high scalability during design time slight modifications have been implemented. The nodes are not only connected to their direct neighbors but also to their neighbors lying on the diagonal, causing a shorter diameter of the whole network, as can be seen in figure 1. Compared to a conventional grid, this introduces a higher connectivity and a shorter diameter.

1.2 Structure of each switch box

When the topology of the network-on-chip is determined, each switch box of the net has to handle the information streams, which are split into several flits by the processor array. Thus, the switch box has to handle each flit independently and autonomously. One task provided by this node is to store the flits. Moreover, in order to relieve the connected processor array the switch box has to route the flit through the net autonomously. This requires a routing decision inside each box and a communication structure between the input and output ports of every node. Figure 2(b) shows the structure of our approach. Here, due to a generic description the whole number of input and output ports



(a) switch box connectivity to other nodes (b) internal structure of each switch box providing one extra port for the connected processor array

Figure 2. Structure and connectivity of each switch box in the middle of the grid

can be changed during design time easily and all components are adapted to the number of ports.

One requirement of each box is to store the flits. Due to independent information streams, a distributed memory approach is used in contrast to shared memory by implementing FIFOs in each port. FIFO structures are able to achieve a higher performance and, compared to shared memory, do not need a complex control structure. In contrast, the FIFOs in each port establish a queuing line and the head-of-line-blocking problem has to be faced if only one queue is provided [3]. Thus, in every input port one queue of FIFOs is used for each output, known in literature as *virtual-output-queuing* (VOQ) [7]. Inside the output ports no parallel queue has to be established since this function can be transferred to the input port of the adjacent switch box.

For the communication structure inside each switch box, a crossbar is established in order meet the high performance requirements inside each box. A bus structure is not able to guarantee high throughput because one participant blocks all the other accesses ever though these might be disjunctive. Anyway, at each output port an arbiter must be provided because two or more inputs could deliver their flits to the same output. A suitable solution to arbitrate these accesses is to provide a round-robin arbiter for each output. Our approach uses a modification of [2] to improve the operating speed. Thus, only two of the proposed arbitrary stages are established with the result of a non-bijective mapping. This leads to a pipeline length of two stages and, consequently, a shorter latency.

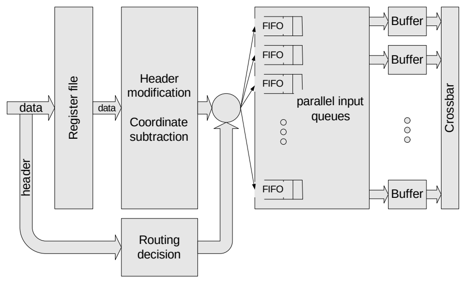


Figure 3. Structure of an input port

1.3 Structure of one port

As mentioned earlier, each switch box has to store each flit at the input or output in a virtual output queue. This task can be managed by an input or an output or by both. Consequently, the function of an output of one node can be transferred to the input of the next neighbor so that complexity is reduced. It suffices to use the storage function in the output port if the interconnection delay between two switch boxes becomes too long. In contrast, if only one storage queue is provided at the output port, head-of-line-blocking will occur also. Thus, nearly the same input structure must be used at the output. So long as the critical path is not determined by the interconnection delay, the output of the crossbar can be directly connected to the input of the neighboring node.

Besides storing the flits and separating them into parallel input queues every input has to change the header of a flit that contains information about its destination. Thus, the structure shown in figure 3 is developed. Here, the flits are stored in a register file because of the routing decision, which takes computation time. In parallel to this decision the header modification is applied and each flit is stored in a FIFO. Multicast and broadcast functions can also be implemented.

2. Routing strategy

In our approach the information is split into several flits that are routed through the net to the target node. In contrast to a circuit-switched network the path through the net is not reserved and each node handles the flits autonomously. Two different strategies can be used to determine the path through the network. Due to the underlying grid topology, Cartesian coordinates can be used to determine the destination node. Each flit consists of a header that includes a two-dimensional destination vector. This vector relatively points to the target node from the current position in the network. Thus, the vector has to be modified if one flit is sent to a neighbor node.

2.1 Dynamic routing

The flits have to be delivered from the source node to its destination. Instead of establishing a central routing decision at the initial node, the dynamic routing is used inside each switch box, especially inside each input port (cf. fig. 3). Central routing algorithms such as Dijkstra or Bellman-Ford algorithms require a global knowledge of the network. This would result in extensive hardware if this task is transferred into the switch box relaxing the processor array. Moreover, these algorithms need information about the communication costs, which are mainly based on the actual traffic in a system-on-chip design since the network and the communication structure between two nodes are regular. Thus, due to high throughput these costs change immediately. This results in another optimal path through the network.

The connectivity of each switch box can be increased (see section 1) resulting in several output ports. Here, each output port is identified by its coordinates and is allocated by costs stored in registers. Each flit is temporarily stored at the input and the information about the header is extracted. With this information, the corresponding quadrant is chosen so that we have a subset of all outputs. The output port with the minimum cost is selected. Then, the flit is stored in the according FIFO queue, the header modification is performed and, with this, the relative position of the flit changes.

2.2 Static routing

By using this strategy the way of each flit through the network is predetermined. When arriving at an input port, the header information is extracted and the flit is switched to the output port depending on the quadrant (x-y routing). For a switch box with a connectivity of four, this results in a transmission to positive or negative x/y direction. First, the corresponding column is chosen by switching the flit to the node at which the x coordinate of the vector equals zero. As long as the y coordinate is unequal to zero the flit is switched in y direction. After storing the flit the header modification is the same for both routing strategies.

This technique chooses the shortest way through the net if a grid with a connectivity of four is used as topology. If the connectivity increases this technique is inefficient and must be adapted. If the x coordinate or y coordinate has already been reached the flit is sent into the remaining y direction or x direction. If no corresponding row or column is reached the next node is chosen depending on the ratio of both coordinates. Thus, the flit covers the largest distance toward its destination so that the shortest way through the net will also be found with these modifications when the connectivity is bigger than the standard connectivity of a grid.

Both possibilities (dynamic and static routing) are implemented in our system. After a flit has been received, it is stored in one of the parallel queues based on the static or dynamic routing. Depending on application demands the routing strategy can be changed immediately by special control flits. Before storing the flit, the header has to be modified due to a change in the position because no storage elements are being provided at the outputs. Thus, the destination vector has to be modified before leaving the queue. Additionally, a header modification behind the queues would have increased the critical path substantially.

3. Synthesis of one node

In this section we analyze the on-chip network related to latency, throughput, area, and power consumption, in dependence on storage capacity and connectivity of the switch box. A first prototype has been implemented on a modern standard cell technology provided by UMC (0.13 μm@1.2 V). Our approach for the communication structure is designed in such a way that several parameters such as flit width, storage capacity of the FIFO structures and the header size and, with this, the whole dimension of the mesh can be easily changed before synthesis.

Table 1 gives an overview of the synthesis results for a different storage capacity and connectivity of each switch box while the width of one flit is set to 32 bit including an eight bit destination vector (although these parameters can be changed later). The connectivity is changed by increasing the number of ports. The table shows the critical path delay, the area of one node, and the power consumption at a certain connectivity. The power is computed with Synopsys Design Power for the maximum speed of each version. The width of the destination vector is fixed because the expected size of the system will be in this dimension. Due to the sign-magnitude representation of the vector a $2^3 \times 2^3$ grid[3] can be addressed, which is suitable for the proposed multiprocessor.

It can be concluded from table 1 that the dynamic routing has a negative impact on the network due to a longer critical path and higher area consumption. Thus, the implementation of both routing strategies has to be considered in dependence on the application. If only static routing is provided a compact design with less area consumption and higher throughput, based on a higher system clock, is achieved, whereas dynamic routing can relieve edges of traffic with the help of cost tables (e.g., in an universal coprocessor design). Furthermore, the proposed network can handle a throughput of 20 Gbit/s per port in each direction if a flit size of 32 bit is assumed. Throughput can be enhanced if the number of parallel interconnections is further increased by using a larger flit size.

Table 1. Results of synthesis on UMC 130 nm @1.2 V (one switch box)

number of flits[1]	**ports per switch box**	**delay bc/wc**[2] **[ns]**	**frequency [GHz]**	**area** $[\mathrm{mm}^2]$	**power bc/wc [W]**	**throughput bc/wc [Gbit/s]**
			dynamic and static routing			
5	5	0.80/1.78	1.25/0.56	0.5621	2,966/0.855	200/90
10	5	0.86/1.85	1.16/0.54	0.9019	2.954/0.834	186/86
20	5	0.90/1.80	1.11/0.55	1.8059	2.886/0.930	177/89
5	9	1.00/2.25	1.00/0.44	1.8488	2.799/0.745	288/128
10	9	1.09/2.30	0.92/0.43	2.9209	3.101/0.803	264/125
20	9	1.13/2.46	0.88/0.41	5.5675	3.579/0.859	255/117
5	17	1.50/3.10	0.66/0.33	6.6744	2.805/0.756	363/175
			static routing only			
5	5	0.72/1.63	1.39/0.61	0.5542	3.368/0.932	222/98
10	5	0.77/1.57	1.30/0.64	0.8803	3.185/0.990	208/102
20	5	0.75/1.56	1.33/0.64	1.7329	3.469/1.045	213/102
5	9	0.95/2.00	1.05/0.50	1.7827	2.901/0.857	303/144
10	9	0.92/2.00	1.08/0.50	2.8223	3.411/0.931	313/144
20	9	0.99/2.10	1.01/0.47	5.5881	3.886/1.024	291/137
5	17	1.10/2.50	0.91/0.40	6.5350	2.804/0.942	494/218

4. Conclusion

In this work, a network-on-chip for an on-chip multiprocessor design has been proposed. The topology was chosen as a grid to provide a regular communication structure and a good re-use possibility for further implementations. Moreover, each node has been designed to handle the traffic autonomously and provides a good scalability with growing networks. Two routing strategies have been implemented in the switch box design and are analyzed with respect to their performance and resource requirements. Providing a maximum throughput of about 500 Gbit/s, a prototypical implementation has been designed by using a 130nm CMOS technology.

Acknowledgment

This work was supported in part by the Infineon Technologies AG, especially the department CPR ST, Prof. Ramacher and by the German Research Council DFG, SFB 376.

Notes

1. stored in each FIFO
2. best case and worst case conditions
3. Thus a $64 \cdot 4 = 256$ processor array can be implemented.

References

[1] A. Brinkmann, J.-C. Niemann, I. Hehemann, D. Langen, M. Porrmann, and U. Rückert. On-Chip Interconnects for Next Generation System-on-Chips. In *Proc. of the 15th Annual IEEE International ASIC/SOC Conference*, pages 211 – 215, September 2002.

[2] A. Gupta, F. G. Gustavson, M. Joshi, and S. Toledo. Design and implementation of a fast crossbar scheduler. *ACM Transactions on Mathematical Software*, 24(1):74–101, 1998.

[3] M. J. Karol, M. G. Hluchyj, and S. P. Morgan. Input versus output queueing on a space-division packet switch. *IEEE trans. on commun.*, COM-35:1347–1356, 1987.

[4] S. Kumar. On packet switching networks for on-chip communication. In A. Jantsch and H. Tenhunen, editors, *Networks on Chip*, chapter 5. Kluwer Academic Publishers, 2003.

[5] D. Langen, J.-C. Niemann, M. Porrmann, H. Kalte, and U. Rückert. Implementation of a risc processor core for soc designs fpga prototype vs. asic implementation. In *Proc. of the IEEE-Workshop: Heterogeneous reconfigurable Systems on Chip (SoC)*, Hamburg, Germany, 2002.

[6] J.-C. Niemann and et al. A holistic methodology for network processor design. In *Proc. of the Workshop on High-Speed Local Networks held in conjunction with the 28th Annual IEEE Conference on Local Computer Networks*, pages 583 – 592, Oct. 2003.

[7] Y. Tamir and G. Frazier. High-performance multiqueue buffers for VLSI communication switches. *15th Annual International Symposium on Computer Architecture*, pages 343–354, 1988.