A Reconfigurable Ethernet Switch for Self-Optimizing Communication Systems

Björn Griese and Mario Porrmann
Heinz Nixdorf Institute and Institute of Electrical Engineering and Information Technology
University of Paderborn, Germany
{bgriese, porrmann}@hni.upb.de

Abstract. Self-optimization is a promising approach to cope with the increasing complexity of today's automation networks. The high complexity is mainly caused by a rising amount of network nodes and increasing real-time requirements. Dynamic hardware reconfiguration is a key technology for self-optimizing systems, enabling, e.g., Real-Time Communication Systems (RCOS) that adapt to varying requirements at runtime. Concerning dynamic reconfiguration of an RCOS, an important requirement is to maintain connections and to support time-constrained communication during reconfiguration. We have developed a dynamically reconfigurable Ethernet switch, which is the main building block of a prototypic implementation of an RCOS network node. Three methods for reconfiguring the Ethernet switch without packet loss are presented. A prototypical implementation of one method is described and analyzed in respect to performance and resource efficiency.

1 Introduction

A prerequisite for the realization of self-optimizing systems is the availability of a hardware infrastructure that is able to adapt to changing application demands at runtime. Traditionally, embedded real-time systems have been designed in a static manner for pre-assigned hardware platforms [1]. The hardware in these systems is fixed and flexibility is only provided by software. In contrast to this we use dynamically reconfigurable hardware, which offers an additional degree of flexibility due to its ability to change the hardware structure at runtime [2]. In the context of dynamically reconfigurable real-time systems [3] new methods have to be developed to meet the real-time requirements in particular during the reconfiguration process.

This work was developed in the course of the Collaborative Research Center 614 – Self-optimizing Concepts and Structures in Mechanical Engineering – University of Paderborn, and was published on its behalf and funded by the Deutsche Forschungsgemeinschaft.

Please use the following format when citing this chapter:

Griese, B., Porrmann, M., 2006, in IFIP International Federation for Information Processing, Volume 216, Biologically Inspired Cooperative Computing, eds. Pan, Y., Rammig, F., Schmeck, H., Solar, M., (Boston: Springer), pp. 115–124.

In this paper we introduce reconfiguration methods for a Real-Time Communication System (RCOS). Our aim is to develop an RCOS for self-optimizing mechatronic systems that efficiently uses the available hardware resources. Self-optimizing systems are able to adapt automatically to dynamically changing environments and user requirements. For an efficient use of the resources, the tasks that control the actors and observe the sensors are distributed to appropriate computing nodes of the system. As a consequence, the real-time requirements and the communication requirements vary as the distribution of tasks changes in the considered mechatronic systems. In order to enable the adaptation to changing environments reconfiguration from the application level down to the hardware level is a required key technology.

The basis of our RCOS is formed by network nodes that allow setting up line and ring topologies. Each node consists of at least two network interfaces that connect the node to its neighbors and to an embedded processor. In order to be able to adapt the network nodes to changes in protocols and interface requirements, which can not be foreseen, we use an FPGA for the implementation of the network interfaces. The architecture of the reconfigurable RCOS network node is shown in Fig. 1.

The RCOS node handles two different types of data streams: data originated from or terminated at the processor and streams that are simply passed through. If network traffic is rather small or if real-time requirements are low or even nonexistent, comparatively simple network interfaces are sufficient, which occupy only a few resources. In this case, data packets are forwarded from one port to another by a software implementation on the embedded processor. This causes a high load for the processor, the internal bus and the memory while the FPGA resources can be utilized by other applications. If the software implementation is not able to deliver the required performance, e.g., due to increasing bandwidth or real-time requirements, the two separate interfaces are substituted by a single integrated hardware switch during runtime. This switch is able to forward data packets autonomously and, as a consequence, manages a much higher amount of traffic. However, the structure of this switch is more complex and requires additional FPGA resources, which are no longer available for other applications.

The idea to use reconfigurable logic for the integration of network applications into the network interface has been realized, e.g., by Underwood et al. [4]. Comparable network interfaces have been used for server and network applications, e.g., web servers, firewalls [5], and virus protection [6]. If the RCOS is implemented in an FPGA, packets that are stored in the active switch implementation may be lost if the switch is overwritten with a new one. Real-time requirements can only be supported if no packet loss is caused by the reconfiguration process. Therefore, three methods for the reconfiguration of RCOS network nodes without packet loss will be introduced in Section 2. Prerequisites for these methods are a packet based communication protocol and the possibility to implement both the software switch and the hardware switch simultaneously on the FPGA during reconfiguration. The RCOS network node is prototypically implemented by a dual-port Ethernet switch. Switched Ethernet technology is commonly used in real-time communication networks in the area of industrial automation, e. g., Industrial Ethernet [7] and PROFINET [8].

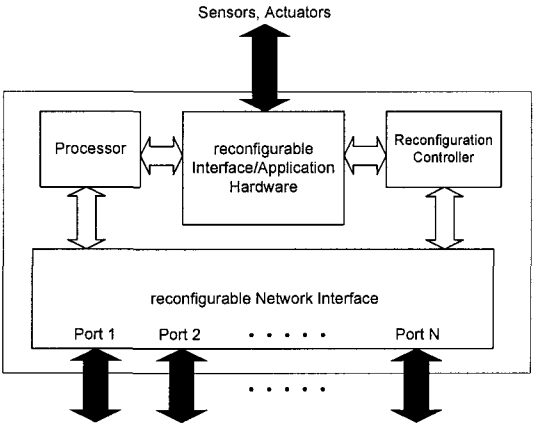

Fig. 1. RCOS network node

A prototypical implementation of our switch has been realized with our RAPTOR2000 System, a Rapid Prototyping System developed at the System and Circuit Technology research group at the University of Paderborn [9]. Xilinx Virtex-II Pro FPGAs are used, which comprise two embedded PowerPC processors in addition to fine-grained reconfigurable hardware resources. The Ethernet switch is implemented either in software or in hardware, as detailed in Section 3. For the prototypical implementation, the reconfiguration is triggered by the network load, which is continuously measured.

2 Strategies for Reconfiguration

The basic principles of dynamic reconfiguration of network interfaces without packet loss have been presented in [4]. Based on this theoretical approach, we present three reconfiguration methods and analyze their practical relevance. To avoid packet loss for all of the three alternatives, both the software switch and the hardware switch have to be active during reconfiguration in order to maintain connections to neighboring nodes. Furthermore, the transmit and receive processes have to be switched separately. As depicted in Fig. 2 the access to the shared Media Independent Interface (MII) is controlled by a hardware multiplexer. If currently no Ethernet packet is received the hardware switch is allowed to switch over the ports from one switch to the other switch. Hence, we use the Inter-Frame Gap (IFG) of the Ethernet protocol to hand over interface control.

In our first approach, the hardware multiplexer immediately switches the signals of the receive process to the "new" configuration if the reconfiguration is started within an IFG. Subsequently, received Ethernet packets are written into the receive buffers of the "new" configuration. Ethernet packets, which are still in the "old" configuration, must be copied to the transmit buffers in the "new" configuration to terminate the reconfiguration of the receive process.

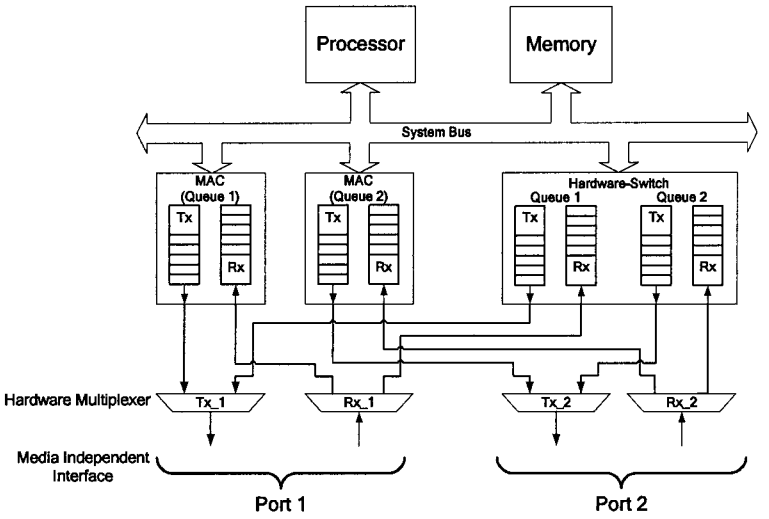

Fig. 2. Architecture of the reconfigurable switch

The transmit process of the "old" reconfiguration transmits all Ethernet packets residing in the send buffer before the hardware multiplexer can switch over the transmit signals. Due to the Ethernet standard, the transmit process has to keep a minimal IFG between two packets, even during the reconfiguration process. Therefore, the duration of a minimal IFG must elapse since the last transmission of a packet, before the signals are switched to the "new" configuration.

A rearrangement of the packet sequence can occur, if the hardware switch forwards a "new" packet before the copy operation from the software switch is finished. But the benefit of this method is a fast activation of packet forwarding by the hardware switch. Hence, no buffer overflow can occur during reconfiguration.

An alternative reconfiguration method is to forward the packets stored in the receive buffer of the "old" configuration to the transmit buffer of the "old" configuration by software. New arriving packets are directed to the hardware switch. The hardware multiplexer switches the transmit signals only when both the receive buffers and the transmit buffers become empty. Concerning the reconfiguration speed, this method is equal to the method described above, because only the write destination has changed. This method has a drawback if the processor is not able to forward all packets with the maximum data rate. The switching of the transmit signals to the "new" configuration is delayed by the copy operation. In this case, the hardware switch is not able to forward new arriving packets, because the transmit signals are still blocked. The risk of a buffer overflow in the hardware switch is present. Larger receive buffers can compensate for this risk. The advantage of this method is that no rearrangement of the packet sequence can occur. As a third alternative, the transmit signals can be switched immediately to the "new" configuration. Therefore, the processor move the content of the transmit buffer of the software switch to the transmit buffer of the hardware switch. This method allows a

fast activation of the transmit process of the "new" configuration. The processor has to verify if receive and transmit buffers are empty. If necessary, the processor copies the packets to the corresponding transmit buffers, requiring additional processing time. The processor has to flush the transmit buffers first, to avoid a rearrangement of the packet sequence. In contrast to the second method that has been described, a rearrangement during the reconfiguration process is possible in this approach. Using this method, no status information must be given to the hardware multiplexer, e.g., no information about the receive and transmit buffer status is required. As described above, the reconfiguration process is finished when the reconfiguration of both receive and transmit processes is accomplished. Maintaining the original packet sequence cannot be guaranteed with the first and with the last presented reconfiguration method. This does not mean that the second method is the only possible solution, but the other methods require packet numbering and reordering functions. The first method allows for a fast activation of the hardware switch, but a rearrangement of the packet sequence is possible. The second method avoids a rearrangement of the packet sequence, but the reconfiguration process is delayed by the copy operations. The third method can be used if no status information of the MACs (Media Access Controller) is available. In this case, the processor is responsible for redirecting the remaining packets, thus decreasing the reconfiguration speed. In our prototypical implementation we used the first method to demonstrate a reconfiguration without packet loss. Due to performance issues of the embedded processor, and in order to achieve a minimum buffer size, our application requires a fast activation of the hardware switch.

3 Implementation of a dynamically reconfigurable Ethernet switch

The dynamically reconfigurable Ethernet switch consists of a software switch and a hardware switch. The software switch comprises two Ethernet Media Access Controllers that are connected to the system bus of the SoPC. For each port of the switch, one Ethernet MAC is required. The switching decision is made by the processor. I.e., the processor checks the destination address field of the Ethernet packets and copies the data from the receive buffer into the transmit buffer, if the packet has to be forwarded to another Ethernet port. The whole Ethernet traffic is transferred via the system bus. The hardware switch has the same capabilities as the Ethernet MACs. In contrast to the software switch, the switching decision is carried out in hardware. Therefore, the processor and the system bus are released from performing network infrastructure tasks.

For being able to implement the proposed RCOS we have developed an Ethernet MAC for Xilinx FPGA, which supports dynamic reconfiguration without packet loss. Therefore, the Ethernet MAC has to generate status information for the processor (e.g., network load and buffer state). In order to maintain the IFG between two Ethernet packets during the reconfiguration process, the transmit process has to be suspended. Additionally, packets already queued in the Ethernet MAC have to be

prevented from being transmitted before the multiplexer has switched the transmit signals to the Ethernet port.

In addition to a software implementation of the Ethernet-switch, a hardware switch has been implemented, which relieves the processor of network infrastructure tasks. In order to avoid the transmission of corrupt packets, the hardware switch uses the store-and-forward switching method. It consists of two independent cross-connected sub-switches, which are connected to the system bus. One sub-switch component is able to forward packets in one direction. The sub-switch component is an extension of the Ethernet MACs presented before. It integrates an additional buffer for receiving and forwarding the packets that are not destined for the processor. Because of the hierarchical structure of our switch, the Ethernet software driver for both the software switch and the hardware switch are the same. For a reconfiguration of the switch no adaptation of application software is necessary.

Reconfiguration Options

To cope with the real-time requirements, the reconfigurable switch is able to collect information that is relevant for quality of service managment. Based on this information the Ethernet switch can be reconfigured. One of the measured parameters is the network load, which is determined by the MACs in our hardware implementation. The length of an Ethernet packet varies between 72 Bytes and 1526 Byte (8 Byte Preamble and Start-of-Frame-Delimiter included). But the measurement of the network load should not depend on the packet size, as far as possible. This can be achieved by measuring the time the data-valid signal (RX_DV) of the MII interface is active in a predefined interval, because RX_DV determines the beginning and the end of an Ethernet packet. Thus, the network load is calculated according to equation (1).

$$L = \frac{K \cdot IFG_{\min} + RX_{active}}{T} \tag{1}$$

L is the network load in percent, where 100 % network load corresponds to the maximum data rate. The factor K represents the number of IFGs, occurring in the measurement interval T. IFG_{min} represents the minimal IFG (0,96 µs for 100 MBit/s) of the applied Ethernet protocol. *RXactive* is the time, in which the RX_DV signal is set. This way of measuring the network load is independent on the packet size. In our prototypical implementation the reconfiguration from the software switch to the hardware switch is started if the network load exceeds 20 %.

Another possibility to maintain the real-time requirements is to reconfigure the Ethernet switch depending on the buffer state. Therefore, an impending buffer overflow is signaled to the processor by the MAC. In this case, the processor initiates a reconfiguration from the software to the hardware switch. Because the buffer is able to save further incoming packets during the reconfiguration process, packet loss is avoided if the reconfiguration is fast enough.

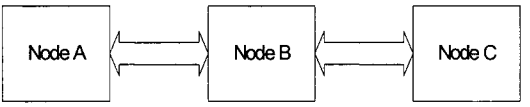

Fig. 3. Demand for low latency

A third alternative to optimize the efficiency of the communication system is to adapt the forwarding latency caused by the switch. Fig. 3 shows a typical situation in an RCOS. Node A, B and C are connected in a line topology. If Node A demands for a low latency to Node C, Node A sends a message to Node B. Node B initiates its reconfiguration and is now able to offer low forwarding latency. While the first two approaches that have been described above perform their optimizations based on local information, the third approach enables a self-optimization of distributed systems based on an interaction of the network nodes.

4 Performance evaluation

The latency of both switch configurations has been analyzed by means of a measurement circuit integrated in the switch implementation together with a network load generator. The network load generator has been configured to send packets with varying packet length to evaluate the correlation between packet size and latency. The values for the packet size given in this paper are related to the data field of the Ethernet protocol. The complete packet size can be calculated by adding 26 Bytes for the Ethernet header.

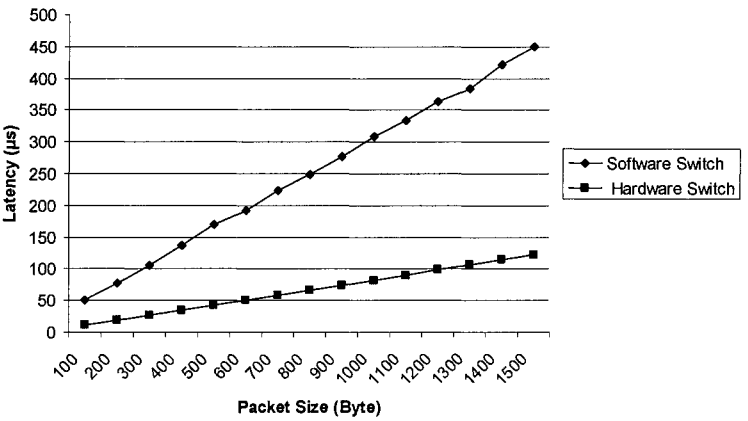

Fig. 4. Measured latency of the software switch and of the hardware switch

Because both the hardware switch and the software switch are store-and-forward implementations, the latency of both switches linearly depends on the packet size, as shown in Fig. 4. But the gradient of the latency of the software switch is 3.92 times higher than the gradient of the latency of the hardware switch. The main reason for the higher gradient is that the processor copies the Ethernet packets from one MAC to memory and from memory to the other MAC in the software implementation. The perturbation in the linearity of the latency graph is caused by processor operations that require a variable number of clock cycles, e.g., due to fetching instructions from external memory or from the instruction cache. Because of the high latencies, the software switch achieves in the worst case (i.e., for minimum packet size) a maximum data rate of 18 MBit/s, i.e., no packet is lost up to this data rate.

In contrast to the software switch the hardware switch supports the full data rate of Fast Ethernet (100 Mbit/s). In this implementation the perturbation in the latency graph is due to the integrated hardware monitor, which observes two buffers sequentially: one buffer for the processor queue and one for packet forwarding. If the hardware switch observes the buffer of the processor while a packet is "ready" to be forwarded, the latency increases by one clock cycle.

The easiest way to dynamically reconfigure from the software switch to the hardware switch is just to overwrite the first switch with the second one. Obviously, during reconfiguration no packets can be forwarded in this case. Reconfiguring the FPGA for exchanging the switches takes about 10 ms. During this reconfiguration period no communication via the switch is possible. In the worst case this results in a loss of more than 1400 packets. This problem can be solved by using both switches in parallel and by managing the reconfiguration with one of the three proposed methods.

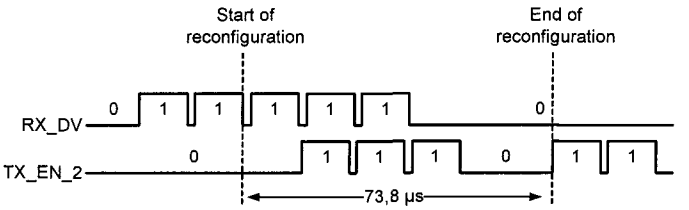

Fig. 5. Illustration of the packet flow during reconfiguration

For testing the proposed approach that uses both switches in parallel during reconfiguration, the reconfiguration has been initiated by forcing an impending buffer overflow in order to verify a lossless reconfiguration. In this scenario, the reconfiguration is initiated if two packets reside in the receive buffer. Five packets with a size of 100 Bytes were sent to the switch, as shown in Fig. 5. The RX_DV signal indicates incoming packets; the TX_EN_2 signal indicates the outgoing packets. The first marker Fig. 5 represents the start and the second marker the end of the reconfiguration process. We have measured a time of 73.8 µs for the reconfiguration process (i.e., the time between marker one and two). In this time 7 packets can reach the switch and have to be buffered or have to be forwarded.

Because at least one switch is always active, no packets are lost during reconfiguration.

Table 1 shows the resource utilization of the software switch, of the hardware switch and of both the hardware switch and the software switch during the reconfiguration process on the Virtex-II-Pro-20 FPGA in terms of required Slices and BlockRAMs. The results have been obtained by synthesizing the design with Xilinx ISE 6.1. The packet forwarding of the software switch is executed by one of the two embedded PowerPC processors. The software switch comprises 24 kByte of Block RAM while the hardware switch occupies 36 kByte due to an additional buffer, which is required for communication with the processor. Therefore, the switch requires 60 kByte for buffering during reconfiguration (when both switches are active).

Table 1.: Resource requirements of the switch implementations

Parameter	**Software switch**		**Hardware switch**		**Switch during reconfiguration**	
Resources	Used/Total	%	Used/Total	%	Used/Total	%
Slices	*2523/9280*	*27*	*6640/9280*	*71*	*9167/9280*	*98*
RAM16	*44/88*	*50*	*52/88*	*59*	*64/88*	*72*

As expected, the high performance of the hardware switch (in terms of low latency and high bandwidth) comes with the cost of large area requirements on the FPGA. If this high performance is required, there is no alternative to a hardware implementation. But if low bandwidth requirements come together with low or no real-time requirements, a small software based network interface is sufficient. In this case the hardware resources of the switch can be used, e.g., to speed up other applications. In order to support a lossless reconfiguration, it is a must that both switch implementations (hardware and software) are able to run concurrently. Therefore, the required FPGA resources for our system are at least given by the sum of the resources of both switches. During reconfiguration nearly 100 % of the Virtex-II-Pro-20 FPGA Slices are occupied (cf. Table 1). After reconfiguration, 30 % of the FPGA resources are available if the hardware switch is active and about 70 % of the reconfigurable logic is available if the software implementation is executed by the processor.

5 Conclusion

In this paper we have introduced methods to use the dynamic reconfiguration capability of FPGAs in real-time communication without violating real-time requirements. A dynamically reconfigurable dual-port Ethernet switch has been used as an example for a prototypical implementation of our approach. The Ethernet switch is able to adapt to changing requirements of an application during runtime. A software switch has been developed for minimum resource consumption and a hardware switch has been realized for maximum performance. The Ethernet switch detects increasing communication requirements by continuously measuring the

network load. These mechanisms enable a self-optimization of the communication infrastructure in distributed systems. The implemented reconfiguration methods described in this paper support loss-less packet processing even throughout the reconfiguration, which is necessary to guarantee real-time behavior in communication systems.

6 References

1. Rammig, F.J.: Autonomic Distributed Real-Time Systems: Challenges and Solutions. In: *7th International Symposium on Object-oriented Real-time Distributed Computing (ISORC 2004)*, May 12-14, 2004.
2. Torresen, J.: Reconfigurable Logic Applied for Designing Adaptive Hardware Systems. In: *Proc. of Int. Conference on Advances in Infrastructure for Electronic Business, Education, Science, and Medicine on the Internet (SSGRR 2002W)*, 2002.
3. Carter, A.: Using Dynamically Reconfigurable Hardware in Real-Time Communications Systems, University of York, November 2001
4. Underwood, K.D.; Sass, R.R.; Ligeon, W.B.: A Reconfigurable Extension to the Network Interface of Beowulf Clusters. In: *Proceedings of the IEEE Conference on Cluster Computing (Cluster 2001)*, 2001.
5. Friedman, D.; Nagle, D.: Building Firewalls with Intelligent Network Interface Cards. Technical Report CMU–CS–00–173. CMU, May 2001.
6. Lockwood, J.W.; Moscola, J.; Reddick, D.; Kulig, M.; Brooks, T.: Application of Hardware Accelerated Extensible Network Nodes for Internet Worm and Virus Protection. In: P*roceedings of the International Working Conference on Active Networks (IWAN)*, 2003.
7. Furrer, F.: Ethernet TCP/IP for industrial automation. Huethig, 1998.
8. PROFIBUS Working Group: PROFInet: Architecture Description and Specification Version 2.01, August 2003.
9. Kalte, H., Porrmann, M., Rückert, U.: A Prototyping Platform for Dynamically Reconfigurable System on Chip Designs. In: *Proc. of the IEEE Workshop Heterogeneous reconfigurable Systems on Chip*. Hamburg, Germany, 2002.
10. Vonnahme, E., Griese, B., Porrmann, M., Rückert, U.: Dynamic reconfiguration of real-time network interfaces. In: *Proceedings of the 4th International Conference on Parallel Computing in Electrical Engineering (PARELEC 2004)*. Dresden, Germany, 7 - 10 September 2004.