Priority-based packet communication on a bus-shaped structure for FPGA-systems

Oliver Sander, Benjamin Glas, Christoph Roth, Jürgen Becker, Klaus D. Müller-Glaser
Institute for Information Processing Technology (ITIV)
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
Email: {sander,glas,roth,becker,kmg}@itiv.uni-karlsruhe.de

***Abstract*—We present an application tailored packed-based SoC communication system with one-hop communication between all entities, priority-based arbitration, broadcast and multicast support on a bus-shaped basis. It is located as a hybrid between NoC and bus approaches, closing the gap for mostly streaming-based systems with the need for highly flexible communication patterns and multicast messages that are below a certain size. The system is implemented and evaluated on a FPGA within a car-to-car communication gateway application.**

I. Introduction

Recent years showed a lot of research in the field of packed-based Network-on-Chip (NoC) communication structures striving to get power-efficient, performant and above all scalable information interchange for future systems. Many comparisons like [1], [2] show that NoCs outperform bus systems above a certain size-threshold. In large communication systems they give benefits regarding performance, power consumption and area used.

But when looking at smaller systems and a more streaming-based computation, a flexible combination of packet-based communication over a specially designed structure can offer high performance with less resource usage than needed for a relatively complex NoC. Based on the car-to-car communication application introduced in [3] we present a streaming-oriented communication structure that offers flexibility, simple interfacing, high throughput and very small latencies using a packed-based communication approach on a bus-shaped structure.

The remainder of this paper is organized as follows: In section II we summarize a selection of related work. Section III motivates our communication approach that is presented in section IV in some detail. In section V we give figures about our implementation and the resources needed while section VI contains performance data. Section VII concludes and points to further research.

II. Related Work

Looking at SoC communication supporting future designs and applications the idea to use packed-based network structures known from device networks has been exploited intensively. These approaches build on networks of switches and/or routers to deliver network packets from sender to receiver. Basic proposals can be found in Dally and Towle [4] proposing a tile-based NoC approach and Jantsch [5] giving a proposal for a NoC protocol stack. Guerrier and Greiner [6] propose the special NoC architecture SPIN providing figures about scalability and performance and also hardware-wrapper for support of different communication interfaces. An approach combining different bus structures connected with a NoC is presented by Wielage and Goossens [7].

Beside the NoC approach there are proposals to extend or upgrade bus structures to fit future demands. Targeting reconfigurable devices the work presented in [8] summarizes state of the art processor buses and discusses their limitations in comparison to NoCs. In [9] the AMBA bus structure is exchanged by a crossbar switch resulting in higher performance of the system. The migration from bus to crossbar is transparent for the IPs that can directly connect to the crossbar. However the distinctive master/slave classification of system buses remains unchanged. Further bus extensions are proposed in [10] allowing faster access to the bus peripherals by dynamically changing the master to bus connection. Another approach is to segment the bus as shown e.g. in [11], [12]. Here the shorter lines give a reduction in power consumption because of the smaller capacitance and also parallel transmission over different segments is supported.

Several approaches have been designed for FPGA based systems. In [13], [14] different approaches for optimized NoC implementations are evaluated. Building a low overhead NoC on token ring basis is presented in [15]. Very few work is published on interfacing IP Cores and NoCs on FPGAs such as the approach presented in [16] introducing a small overhead NoC OPB connection. Sophisticated communication structures for partial and dynamic partial reconfiguration were presented in [17], [18]

III. Motivation for a tailored approach

We base our considerations on a concrete scenario serving as an example for similar specialized systems. Our application is a car-to-car communication (C2CC) system integrated into a central automotive gateway. The system concept can be found in [3] for the overall system and in [19] for the gateway part. Here we just recall the parts needed to understand the basic system layout. Figure 1 gives a rough overview of the relevant processing units and dataflows.

978-3-9810801-5-5/DATE09 © 2009 EDAA

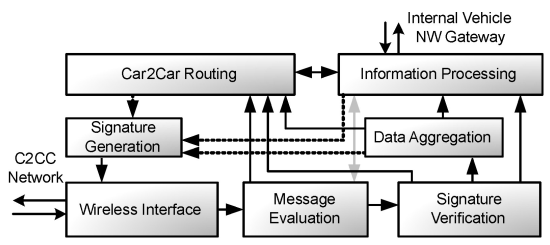

Fig. 1. C2CC gateway system data flow

The depicted C2CC module implements the complete processing of C2C beaconing messages according to the upcoming WAVE standard, focusing on security [20], [21]. It shall be integrated in a FPGA automotive gateway and achieve hardware separation of internal and external network, presenting a clearly defined interface for relevant information between the C2CC network and the internal vehicle network gateway. In special situations (i.e. traffic jam) up to 3000 messages per second from many dozens of vehicles 1 are received and need to be processed and forwarded thus creating a high processing load for the overall system. The internal communication between all modules should impose low latencies and support prioritization of messages to support accelerated handling for important information. Also shall there be no blocking of entities on the same priority level.

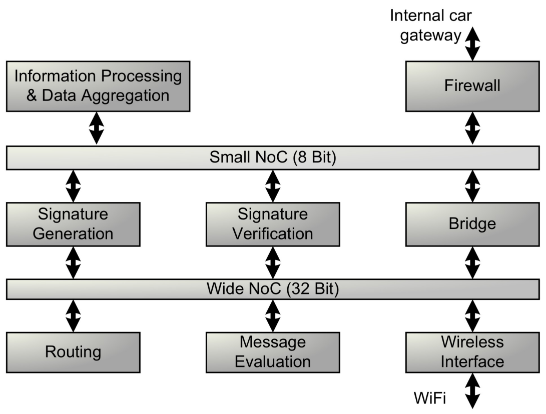

Fig. 2. C2CC gateway system architecture

For the communication structure there are basically three options – direct wiring, bus communication and NoC communication (see section II):

Although the system's operation is dataflow-oriented, no singular dataflow can be determined along which processing units could be chained. Depending on the situation and the beacon contents, different units process the packets and different paths are used. So communication by direct links would create a complex and costly[1] structure.

A classical bus structure has serious drawbacks as well. As all processing units are part of a pipelined processing chain, no partitioning in master and slave modules is apparent. All modules have to be equipped with both master and slave interfaces implicating significant overhead or every transfer from slave-to-slave has to be done over a master thus doubling the bus occupancy time.

[1]in terms of area and resources used

A full-grown NoC finally gives probably the highest throughput for the communication. But this comes along with a complex structure of routers and switches causing hardware overhead and additional issues like routing algorithms. Also some messages in the C2CC module are used by several units. Therefore a multicast functionality is beneficial which complicates the routing further, so that this approach, too, would result in serious overhead and complexity.

IV. System design

Based on these considerations we propose a tailored approach located somewhere in between a classical bus and a NoC, implementing a packet-based communication over a bus structure. We accept the obvious bottleneck of a bus as we have a small number of nodes which is far less compared to the break even of two dozens stated in [1]. All modules are enabled to send and receive. Broadcast and multicast can be realized by addressing groups of modules. For arbitration we chose a multi-criteria scheme with dynamic priorities so that every sender is able to send each packet at any priority level. Packets on the same priority level are scheduled round robin (RR). In the following we look at the different parts in some more detail.

A. Packet protocol

Each packet is divided into header and data field. Because only on-chip-communication is considered, we omit a CRC mechanism. Table I details the header of a data packet. The allocation of the fields is given below.

Header						DATA
Byte0	Byte1	Byte2			Byte3	
7..0	7..0	7..4	3	2..0	7..0	
SRC	DST	PRI	CF	res	LEN	

TABLE I
HEADER STRUCTURE FOR DATA PACKETS

SRC:	8 bit source address, always the address of the sender, consisting of 5 bits node ID and 3 bits subnet mask.
DST:	8 bit destination address. This may be either a module or a group of modules, in the latter case implementing a multicast message. DST 0 is defined as broadcast destination.
PRI:	4 bit priority information. In bit 7..5 application priorities are coded while bit 4 is a superpriority (SP) flag for control messages.
CF:	Chaining-Flag. This indicates the packet is part of a chained multi-packed transmission.
res:	3 reserved bits for future protocol extensions.
LEN:	Data length field in byte.

The maximum data field length of 128 byte is chosen relatively small to minimize the delay for high-priority messages. An arbitrary number of single packets can be chained to a multi-packed transmission enforcing delivery in the correct order and without delivering any other packets to the destination in between.

To optimize transfer volumes and to ensure data and information integrity, the bus is designed to avoid any packet loss. Therefore a blocking mechanism is implemented that is described in section IV-D

B. System layout

The system consists of a single central arbiter with all modules attached to it. Each module has write lines to the arbiter in full bus width. After winning the arbitration they are connected to the central bus via multiplexers. All modules listen to the central bus. Based on the destination address in the packet header they decide whether to store or ignore the packet. To enable multicast messages each module can store several addresses or IDs, one unique ID, and some group IDs. DST 0 serves as broadcast address.

Our test system consists of four nodes as depicted in section VI, figure 8. As the final system consists of five to eight processing nodes, this is sufficient for testing. If necessary several of these systems can be connected via bridges. The 3 address-bits for a subnet mask are reserved for this enhancement. The following paragraphs survey arbiter, module interfaces, and bridge separately.

C. Central arbiter

Arbitration is done by a central module. Send requests are scheduled according to their priorities. Implemented are eight application priority levels and an additional SP level for blocking messages (see IV-A). On each priority level a RR scheduling with history is done, so that after arbitration of a higher priority packet the schedule is resumed at the correct state. As RR is also applied on the highest priority level a guaranteed latency time for such packages can be found. This guarantee is also valid for lower priorities when no request on any upper priority layer exists. Figure 3 gives an overview of the central arbitration unit.

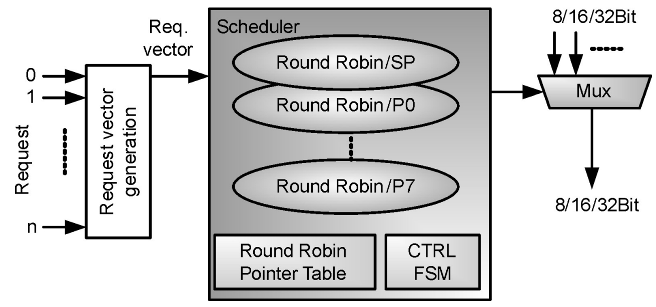

Fig. 3. Structure of the central arbiter

A complete arbitration process is done within one single clock cycle. Out of all requests the highest priority level is detected and a request vector which contains all requests on the selected level are transmitted to the scheduler. A RR pointer contains the node ID of the node that has sent the last packet. Such a pointer is stored for each priority level. The scheduler selects the successor based on the request vector and the pointer information. Bus access is granted by switching the bus multiplexer to the selected node ID. Adapting the arbiter to different bus widths is quite simple as the multiplexer is the only element that depends on the bus width. Currently the arbiter is configurable to 8, 16 and 32 Bit. The FSM monitors overall module behavior as well as correct frame length.

D. Node interface

Packet encapsulation is done by the node interface shown in figure 4. We decided to use a host register interface with a simple processor bus standard and configurable register width of 8,16 or 32 Bit. So we can simply add our interface to a 32 Bit MicroBlaze OPB bus [8] as well as an 8 Bit PicoBlaze port. The packet bus width is configurable to 8,16 and 32 Bit as well. Host interface and packet bus width are independent. Packet headers are generated by the application due to possibly multiple identities of each node. This allows different tasks to be joined on one processing unit.

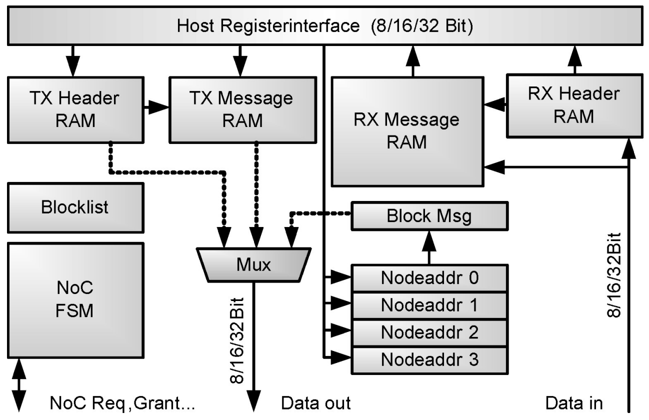

Fig. 4. Structure of the node interface

The TX buffer has space for two packets. So enqueueing a new packet is possible while sending of the prior packet is in progress. Space for buffering multiple packets is implemented in the RX buffer where the queue also respects packet priorities (see below). Packet transmission is organized by the control FSM which requests bus access and transmits the packet to the bus autonomously.

Valid incoming packets with matching ID are stored into the RX buffer. One BRAM block has been used for data storage which is sufficient for 16 packets. Packet loss due to full RX buffers in any of the nodes is not acceptable. To avoid an overflow a special blocking packet with highest priority is sent as soon as the node's buffer is full and the current transmission matches any of the node's IDs. The current packet transmission is interrupted by the arbiter and the blocking packet is broadcasted to all nodes storing the information in their blocking table hindering any further request targeting the critical node. The state remains until the blocking is revoked by the critical node. Any other transmission can be continued. Using this mechanism, acknowledgments are not necessary.

Some application packets must be split into several bus packets that should be stored sequentially in the RX buffer. This buffering constraint avoids fragmentation and overhead for processing within the module. During chaining transmission no other packet shall be received. Therefore a special chaining packet is sent blocking all packet transmissions addressing the chaining destination except for the packet chain sender. The chain block is revoked by the receiver as soon as the chained transmission is completed.

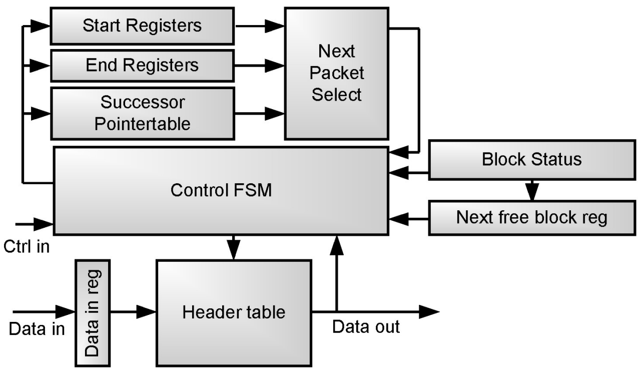

Fig. 5. Structure of the RX header memory

Packet prioritization is a major request within our overall C2CCC application. Therefore a preferred handling of messages in the RX buffer is done additionally to the arbitration prioritization to reduce packet latency of high priority packets. The host interface only sees the packet with highest priority. All packet headers are stored and sorted by the RX Header RAM according to their priority. Packets with identical priority levels are sorted in FIFO style. The information is organized in a linked list running in hardware. The first list element is selected and displayed to the host interface. In addition the address of the first and last header of every priority level are stored. This allows to select, add and remove list elements within a single clock cycle (see figure 5).

E. Bridge

Multiple net segments of different bit widths can be connected via a bridge. Our implementation is configurable freely and independently on both sides to bit widths of 8, 16 or 32. The concept is depicted in figure 6.

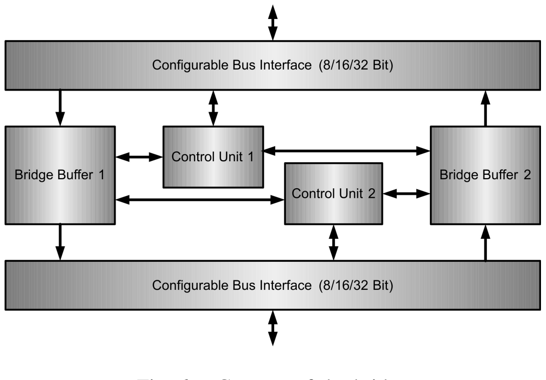
Configurable Bus Interface (8/16/32 Bit)
Bridge Buffer 1
Control Unit 1
Control Unit 2
Bridge Buffer 2
Configurable Bus Interface (8/16/32 Bit)

Fig. 6. Concept of the bridge

The bridge consists of two bridge buffers supporting blocking, chaining and prioritization. The structure is similar to two crosswise connected node interfaces but without a host interface or TX buffer. The RX buffer of the opposite bridge buffer acts as respective TX buffer. Figure 7 gives an overview of the structure. As the bridge is implemented completely in hardware and controlled by a FSM no separate processor is needed for data transport. The latency between end of the read operation and sending request on the opposite side is just one clock cycle. This was approved by a performance test with two connected buses with 8Bit and 32Bit width.

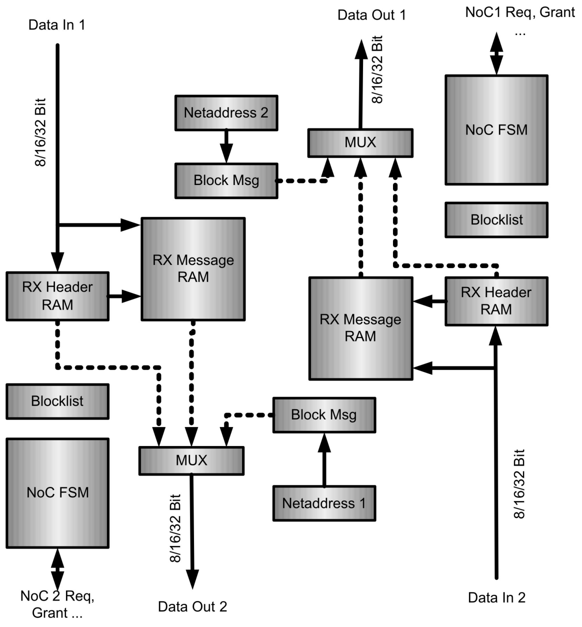

Fig. 7. Structure of the bridge

To enable simple routing between different net segments the node IDs are divided into two parts, a local ID identifying the node in the segment and the global net ID identifying the net segment. The bridge itself is then programmed only to two net IDs. Upon reception of a packet the FSM checks whether the net ID matches the ID of the opposite side and then forwards or drops the packet accordingly.

To connect more then two net segments the bridge has to be extended with a routing table for net IDs to be able to send packets over multiple net segments.

V. Implementation and Resource Usage

The system has been implemented and tested in several versions on various Xilinx FPGAs. Three different bus widths and four numbers of priority levels were used. The default system has a bus width of 32 bit and eight priority levels plus one SP level. These parameters are used if not stated explicitly otherwise.

We again regard central arbiter and node interfaces separately. Table IIa gives selected results about the arbiter implementation for different bus widths. The maximum frequency displays no dependency on the bitwidth of the arbiter. As expected the arbiter size slightly increases with bus bandwidth solely caused by the multiplexer width.

a) Resource usage central arbiter

FPGA type (Xilinx)	unit	8 bit	16 bit	32 bit
Spartan3	Slices(map)	179	200	235
	Gates	2259	2506	2918
	f_{max}(MHz)	66.37	67.58	66.36
Virtex2P	Slices(map)	178	195	232
	Gates	2235	2485	2894
	f_{max}(MHz)	108.41	108.90	116.41
Virtex4	Slices(map)	180	203	238
	Gates	2280	2527	2945
	f_{max}(MHz)	145.96	148.58	144.84
Virtex5	Slices(map)	246	280	335
	Gates	2051	2288	2665
	f_{max}(MHz)	216.78	216.78	216.61

b) Resource usage node interface

FPGA type (Xilinx)	unit	8 bit	16 bit	32 bit
Spartan3	Slices(map)	720	717	768
	Gates	79870	79849	80463
	f_{max}(MHz)	73.44	79.73	71.04
Virtex2P	Slices(map)	705	710	757
	Gates	84244	84368	84899
	f_{max}(MHz)	136.91	140.12	126.58
Virtex4	Slices(map)	716	744	777
	Gates	79817	80631	80894
	f_{max}(MHz)	165.32	163.88	166.11
Virtex5	Slices(map)	763	770	918
	Gates	82889	82897	149289
	f_{max}(MHz)	252.92	238.53	249.72
All architectures	Mem (kbit)	18	18	18

TABLE II
RESOURCE USAGE OVERVIEW

Central Arbiter on Xilinx Spartan3

unit	32 bit 8 prio	32 bit 4 prio	32 bit 2 prio	32 bit 1 prio	8 bit 1 prio
Slices(map)	235	184	161	149	93
Gates	2918	2330	2060	1871	1185
f_{max}(MHz)	66.37	63.28	71.86	72.17	73.06

Node interface on Xilinx Spartan3

unit	32 bit 8 prio	32 bit 4 prio	32 bit 2 prio	32 bit 1 prio	8 bit 1 prio
Slices(map)	768	698	648	606	548
Gates	80463	79378	78667	78084	77343
f_{max}(MHz)	71.04	71.04	71.04	71.04	90.66

TABLE III
RESOURCE USAGE FOR DIFFERENT NUMBERS OF PRIORITY LEVELS

Table IIb shows results for the node interfaces. The size is fairly large mainly due to the prioritization and blocking mechanisms. Resource usage could be minimized by dropping the prioritization in the buffer and using a simple FIFO. A pragmatic simplification of the RX buffer towards a FIFO gave about 360 Slices on a Spartan 3. Besides the implementation with eight application priority levels, the system was also implemented using less priorities[2]. Table III gives an overview of the changed resource usage of the modified system. A significant resource reduction of 36% can be achieved for the central arbiter with one priority level. Further reduction by reducing the bitwidth gives another 24%. The node interface resource consumption slightly decreases by about 21% with one priority level. Combining the 8 Bit version with one priority level gives an increase of the maximal frequency of 20MHz. In general the node interface size is almost equal for 8 and 16 Bit versions on all architectures. Moreover the size of the node interface strongly depends on the buffer size.

To the best of our knowledge features like hardware prioritization and chaining without dedicated channels can be hardly found in other implementations. For example NoC interfaces do not implement buffer prioritization which covers for a large part of our resource consumption. Often the node interfaces are integrated into routers as simple FIFOs and all together show sizes of approximately $450 - 650$ slices [14], [13].

[2]The SP level is always additional to the given number of application priority levels.

VI. PERFORMANCE EVALUATION

The system was evaluated using the test system shown in figure 8. Four different modules, each containing a soft processor core, are attached to the central arbiter. The MicroBlaze (B) runs a TCP/IP stack that allows for packet sending and receiving from a standard PC with some debugging software. PicoBlaze (D) has a simple UART Interface for text outputs. PicoBlaze (C) can act as a packet source generating as much traffic as possible suitable for performance measurements. To get our structure to its performance limits, we added some hardware packet generation. PicoBlaze (A) sends some standard packets for functional verification. A special test interface was added to the arbiter in order to count packets per second. The output is given to PicoBlaze (D). In addition Chipscope was used for functional verification. On this basis several scenarios were tested to verify the theoretical performance values and the correct functional behavior.

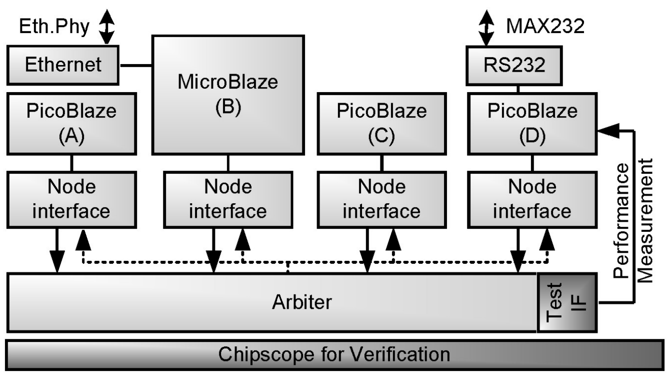

Fig. 8. Setup of the system

The packet protocol (see IV-A) defines a constant header size of four byte and a maximum packet size of 132 byte, consisting of the header and 128 byte payload. At a bus speed of 50 MHz this results in a maximum throughput of almost 1.55GBit/s. Our implementation imposes an additional arbitration gap of four clock cycles per packet, reducing the throughput to 1.4GBit/s by loosing four times the bitwidth per frame. Closing this arbitration gap is one goal for further improvement.

Property	unit	8 bit	16 bit	32 bit
Channel capacity (cc)	MBit/s	400	800	1600
Header transmission	clk cycle	4	2	1
Arbitration gap	clk cycle	4	4	4
Payload capacity	MBit/s	44.4	57.1	66.7
min. packet size (5 Byte)	% of cc	11.1	7.1	4.2
Payload capacity	MBit/s	376.4	731.4	1383.7
max. packet size (128 Byte)	% of cc	94.1	91.4	86.5
Packet latency, max. size,	clk cycle	136	70	37
immediate arbitration	μs	2.72	1.4	0.74
Max. delay to arbitration	clk cycle	408	210	111
using highest PRI	μs	8.16	4.2	2.22

TABLE IV
TEST RESULTS PERFORMANCE

Testing showed the implementation performs correctly up to the theoretical maximum. Table IV gives some performance values for different bus widths measured on a Xilinx Virtex-II Pro FPGA at a clock speed of 50 MHz.

Latencies were evaluated with all four modules sending on highest priority. Using a bus structure we benefit from one-hop transmission. NoC approaches usually induce per-hop-latencies of 2 [13] to 38 [14] clock cycles. Instead our waiting time may be longer, lacking alternative routes from source to destination.

VII. Conclusion and further research

We presented a packed-based communication system over a central bus structure located in between classical bus approaches and recent NoC solutions. The approach is tailored for small to mid size data-stream oriented systems. All nodes are able to send and receive on arbitrary priority levels with round robin scheduling on each level. Broadcast and multicast messages are supported and packet loss is prevented. The system's performance as well as resource usage were evaluated on various real hardware implementations.

Compared to a classical bus system our approach gives a higher modularization by encapsulating all bus access and most of the protocol issues like chaining and blocking in the hardware interface. This is especially beneficial thinking of hardware modules with only minimal control logic.

In contrast to a NoC our pragmatic approach goes without the internal infrastructure of routers and switches. The underlying bus structure accounts for the relatively high percentage of broadcast and multicast messages coming from the application. We also achieve very small minimal latencies transmitting from sender to receiver in a single hop.

Further research includes optimization of the overall system structure. Also the arbitration gap shall be narrowed thus further increasing performance. The system will then be integrated into a car-to-car communication system.

References

[1] C. Zeferino, M. Kreutz, L. Carro, and A. Susin, "A study on communication issues for systems-on-chip," *Integrated Circuits and Systems Design, 2002. Proceedings. 15th Symposium on*, pp. 121–126, 2002.

[2] H. G. Lee, N. Chang, U. Y. Ogras, and R. Marculescu, "On-chip communication architecture exploration: A quantitative evaluation of point-to-point, bus, and network-on- chip approaches," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 12, no. 3, pp. 1–20, 2007.

[3] O. Sander, B. Glas, J. Becker, and K. D. Müller-Glaser, "An exploitation of reconfigurable hardware architectures for car-to-car communication considering automotive requirements," in *FISITA World Automotive Congress 2008*, Munich, Germany, 2008.

[4] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Design Automation COnference (DAC) 2001*. ACM, 2001.

[5] A. Jantsch, "Nocs: a new contract between hardware and software," *Digital System Design, 2003. Proceedings. Euromicro Symposium on*, pp. 10–16, Sept. 2003.

[6] P. Guerrier and A. Greiner, "A generic architecture for on-chip packet-switched interconnections," in *DATE 2000*, 2000, pp. 250–256.

[7] P. Wielage and K. Goossens, "Networks on silicon: Blessing or nightmare?" in *Proceedings of the Euromicro Symposium on Digital System Design (DSD'02)*. IEEE, 2002.

[8] A. Lee and N. Bergmann, "On-chip communication architectures for reconfigurable system-on-chip," *Field-Programmable Technology (FPT)*, pp. 332–335, Dec. 2003.

[9] S. Lee, C. Lee, and H.-J. Lee, "A new multi-channel on-chip-bus architecture for system-on-chips," *IEEE International SOC Conference*, pp. 305–308, Sept. 2004.

[10] K. Sekar, K. Lahiri, A. Raghunathan, and S. Dey, "Flexbus: a high-performance system-on-chip communication architecture with a dynamically configurable topology," in *DAC '05*. New York, NY, USA: ACM, 2005, pp. 571–574.

[11] T. Seceleanu, "Communication on a segmented bus," *SOC Conference, 2004. Proceedings. IEEE International*, pp. 205–208, Sept. 2004.

[12] W.-B. Jone, J. S. Wang, H.-I. Lu, I. P. Hsu, and J.-Y. Chen, "Design theory and implementation for low-power segmented bus systems," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 8, no. 1, pp. 38–54, 2003.

[13] A. Ehliar, J. Eilert, and D. Liu, "A Comparison of Three FPGA optimized NoC Architectures," in *Swedish System-on-Chip Conference (SSoCC)*, 2007.

[14] B. Sethuraman, P. Bhattacharya, J. Khan, and R. Vemuri, "Lipar: A light-weight parallel router for fpga-based networks-on-chip," in *GLSVSLI '05*. New York, NY, USA: ACM, 2005, pp. 452–457.

[15] K. Hadjiat, F. St-Pierre, G. Bois, Y. Savaria, M. Langevin, and P. Paulin, "An fpga implementation of a scalable network-on-chip based on the token ring concept," *Electronics, Circuits and Systems, 2007. ICECS 2007. 14th IEEE International Conference on*, pp. 995–998, Dec. 2007.

[16] R. Holsmark, A. Johansson, and S. Kumar, "On connercting cores to packet switched on-chip networks: a case study with microblaze processor cores," in *7th IEEE workshop DDECS 04*. IEEE, 2004.

[17] C. Bobda and A. Ahmadinia, "Dynamic interconnection of reconfigurable modules on reconfigurable devices," *Design & Test of Computers, IEEE*, vol. 22, no. 5, pp. 443–451, Sept.-Oct. 2005.

[18] M. Hubner, L. Braun, D. Gohringer, and J. Becker, "Run-time reconfigurable adaptive multilayer network-on-chip for fpga-based systems," *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pp. 1–6, April 2008.

[19] O. Sander, J. Becker, M. Hübner, M. Dreschmann, J. Luka, M. Traub, and T. Weber, "Modular system concept for FPGA-based Automotive Gateway," in *Electronic Systems for Vehicles*, ser. VDI-Berichte, no. 2000, VDI. Verein Deutscher Ingenieure, 2007, pp. 223–232.

[20] IEEE, "Trial-use standard for wireless access in vehicular environments (wave)," *IEEE Std 1609*, 2006.

[21] SAE, "Dedicated Short Range Communications (DSRC) Standard Draft," *SAE Standard J2735*, 2006.