An FPGA Bridge Preserving Traffic Quality of Service for On-Chip Network-Based Systems

Ashkan Beyranvand Nejad and Matias Escudero Martinez
Delft University of Technology, the Netherlands
A.BeyranvandNejad@tudelft.nl

Kees Goossens
Eindhoven University of Technology, the Netherlands
K.G.W.Goossens@tue.nl

***Abstract*—FPGA prototyping of recent large Systems on Chip (SoCs) is very challenging due to the resource limitation of a single FPGA. Moreover, having external access to SoCs for verification and debug purposes is essential. In this paper, we suggest to partition a network-on-chip (NoC) based system into smaller sub-systems each with their own NoC, and each of which is implemented on a separate FPGA board. Multiple SoC ASICs can be bridged in the same way.**

The scheme that interconnects the sub-systems should offer the application connections the required quality of service (QoS). In this paper, we investigate bridging schemes at different levels of the NoC protocol stack. Comparing the distinct design criteria for the proposed schemes, a bridge is designed. The bridge experiments show that it provides QoS in terms of bandwith and latency.

I. Introduction

FPGA prototyping of SoCs has become common for early software development and hardware design verification. Electronic consumer applications demands, and recent advances in chip design technologies, are increasing the size of new SoCs dramatically. One FPGA's capacity, however, is not enough for prototyping the large designs in which recently there has been a large efforts of utilizing scalable on chip interconnection network, which is known as Network on Chip (NoC) [1], instead of the traditional bus paradigm. To prototype a large design, a system is required to be partitioned into number of sub-systems, each of them is implemented on a single FPGA. An off-chip scheme is therefore required to connect these FPGAs in order to emulate the design. Although an FPGA emulation board typically includes one FPGA chip, multi-FPGA boards also exist. Hence, not only a bridging scheme between multiple FPGAs on a single board is needed, but also an FPGA board-to-board bridge (off-board bridge) is essential. Similarly, multi-ASIC systems [2] can be constructed by providing a bridge between the NoCs on each chip.

Furthermore, run-time configuration of the on-chip interconnect for different use cases [4] requires a *host* which is locally responsible for the configuration of the NoC. Having an off-board bridge to an external host such as a Personal Computer (PC), allows the interconnect configuration to be performed remotely. Accordingly, the system verification and debug by accessing from outside of the chip to the on-chip system state and memories contents is become possible.

The unified system resulting from connecting the sub-systems located on different boards, should fulfill the applications requirements, i.e., the applications' connections must receive the appropriate bandwidth and latency (QoS) when crossing both sub-NoCs and the bridge.

In this paper, we propose a bridging scheme between multiple chips (FPGA, ASIC, etc.) that each contain a NoC. There are many possibilities to divide such a system into smaller sub-systems. We explore the various possible bridging schemes by investigating the possible bridge insertion into the interconnect at distinct layers of the protocol stack.

An interconnect protocol stack model for NoC-based systems, which is based on the OSI reference model [5], is proposed in [6]. This model contains five layers referred to as Session, Transport, Network, Data link, and Physical layer. In this work, a proposed bridging scheme at each layer is investigated to support the full decoupling of the FPGA boards, while preserving the application QoS and being cost efficient. For this purpose we introduce six design criteria based on which the schemes are compared.

The rest of this paper is organised as follows. In Section II we review related work. Section III gives an overview of the on-chip interconnection network. We investigate the bridging design space in Section IV followed by the bridge detailed architecture in Section V. The experimental results are presented in Section VI. Finally, section VII concludes this paper.

II. Related Work

Significant research efforts have been done in different aspects of interconnecting separate chips. These aspects regard either the bridging over chips that are located on the same board (on-board bridging) or bridging over different boards (off-board bridging). The interface level of the bridge to the systems on chip is another important issue.

An off-chip NoC interface is proposed in [7]. In this work, the interface is implemented at the physical layer in order to offer a unified view of the NoC protocol to different NoC-based subsystems. This on-board interface has a parallel connection of 78 signals between chips per each bidirectional link.

The concept of sub-NoCs and their interconnection are discussed in [8]. The sub-NoC interconnection is realized at the network layer by a synchronizer module. They can preserve the QoS of connections over on-board sub-NoCs. A

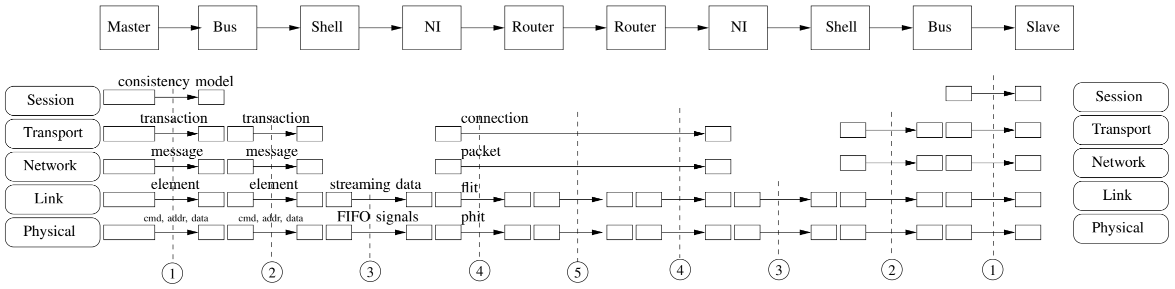


Fig. 1. Overview of on-chip interconnect connection with the involved protocol stack layers at the links

Time Division Multiplexing Access (TDMA) synchronizer is presented, that is in charge of adapting the TDMA slot tables between two sub-NoCs keeping the guaranteed service.

From the aspect of ASIC emulation by utilizing FPGAs, the work presented in [9] shows how a multi-processor SoC (MPSoC) with 48 cores can be fitted in 4 FPGA's by extending a NoC with off-chip synchronous links. A multi purpose emulation platform which can be used with different NoC topologies is also presented in [10], [11]. Network links between routers placed in different chips are emulated by using high speed serial links as both inter-chip and inter-board connections. However, their solution does not provide the QoS management for the connections with different traffic classes.

In [12] a bridge is developed as a NoC interface of a multi-core SoC to the Internet. The bridge operates at the transport layer of the NoC and at the application layer of the Internet side. Multi-processor systems can also take advantage of the intrinsic multi-hop nature of the NoC, that can allow inter-chip connection transparently for the IPs [13] [14] [3]. In [13] the design and implementation of an inter-chip interconnect for 4 DSP's per chip is presented. The inter-chip interconnect module is responsible for the conversion and transmission of data between multiple SoCs using PCI Express as the off-chip connection. The bridge is located at the data link level. Since there are different clock frequencies in the intra-chip connections compared to the inter-chip connections, the bridge is also in charge of accommodating the data rates using an asynchronous buffers.

The companion chip architecture of [2] provides the requirements for an inter-ASIC NoC-based bridge but no architecture or implementation.

All the works mentioned above suffer from lack of either providing off-board SoC interconnection or preserving the QoS of applications of which the connections are bridged. In this paper we bring these two together to propose a generic bridge scheme that manages the QoS of applications, while it connects the SoCs implemented on the different FPGA boards.

III. On-Chip Interconnect Overview

In this section an overview of an interconnection network, to which the bridging scheme is applied, is given. A connection between two Intellectual Property (IP) components is set up via the interconnect. The IPs are illustrated as a *master* and a *slave* in Figure 1.

The interconnect is formed by the traditional *bus* paradigm combined with a NoC. The *master* issues a request (e.g. *write* or *read*) to the bus. The bus is responsible for handling the distributed shared-memory communications in order to send the request to an specific connection *shell* (point 2 in the Figure). The bus communication uses one of the standard interfacing protocols (e.g., DTL [15], AXI [16]). A shell serializes/de-serializes the protocol specific data elements (*Commands, Address, Data*) to/from a Point-to-Point Streaming Data (PPSD) [6] in a handshake-based interfacing protocol at point 3. The PPSD is then packetised/de-packetised by the Network Interface (NI) to/from a network data *packets* from/to flow control digits (*flits*) at point 4. A flit is formed by a sequence of physical digits (*phits*) which are the unit of data transferred in a clock cycle on a *hop* of the interconnect. The connection packets reside in a dedicated NI buffer waiting to be scheduled at an specific time. A Time Division Multiplexing (TDM) scheduler schedules the flits according to its pre-programmed time-slot table in the NI. This implies a timing dependency between interconnect hops, in which the data should arrive in the next hop at a specific time according to the TDM table. The flits are routed by *routers* through the links (point 5) on a path determined in the flit header of a packet [18]. Once the request reaches the destination NI, the above procedure occurs in reverse till the slave accepts the request.

The slave's response follows the same scenario. A connection therefore is formed by a request channel and a response channel. The interconnect deals with the connection traffics differently based on the required service classes. The service provided for a connection data is either Guaranteed Throughput (GT) or Best Effort (BE). In this paper we apply our analysis to NoCs that provide both GT and BE QoS classes, such as Nostrum [17] and Æthereal [18], but it can be equally applied to any NoC that uses connections to provide QoS.

IV. Design Space Investigation

Considering the on-chip interconnect illustrated in Figure 1, there are various physical communication links in the network. It is possible to cut the interconnect at each of these links to insert a bridge in between. This would divide the interconnect into two (or more) sub-interconnects. Moreover, there is more than one layer of the interconnect protocol stack which is involved at a communication link. This would imply that the bridge can be also implemented at different layers of the protocol stack.

A. Bridge Requirements

Having the various design choices, we first present the requirements for the bridge, in order of priority:

1) ***Transparency:*** ideally from an application point of view the bridge should be invisible. This is achievable by having as little manipulation as possible in the travelling data over the bridge. The lower protocol stack layer the bridge is implemented on, the simpler is to design a bridge that is transparent to the applications.
2) ***Decoupling:*** two bridged sub-systems will be implemented on different FPGA boards. The bridge therefore should support the full decoupling [19] between the two systems. Decoupling includes both physical and temporal dependencies, e.g., voltage levels, frequency, location distance, etc. The physical dependency is due to the board-to-board bridging and will be solved by the off-board interfacing protocol. As explained in Section III, the temporal dependency is inside the interconnect and is because of the TDM scheduling scheme for GT traffic.
3) ***Quality of Service (QoS):*** the bridge should preserve the Quality of Service (QoS) of the applications running on the bridged systems, e.g. no GT traffic should wait for any BE traffic to be serviced.
4) ***Cost:*** the cost is both the implementation cost in terms of area (e.g. buffer sizes, number of wires, and size of logic circuit), and the time cost in terms of throughput and latency. The less the implementation cost of the bridge, the more desirable the bridge.

In the rest of this section, we introduce the bridge design criteria that have an impact on the aforementioned requirements. The interconnect protocol stack layers in every possible physical communication links are also explored to realize the most proper bridging solution based on the design criteria, in order to fulfill the requirements.

B. Design Criteria

The design criteria are selected properties of the interconnect links, that can have either direct or indirect impact on designing the bridge architecture, in order to fulfill the aforementioned bridging requirements. The design criteria are distinguished as following:

- *Parallel/Serial Bridge:* A bridge might be implemented as a serial or a parallel link. The parallel link is faster, while it is suitable for a short distance of bridging. However, the serial one is slower and is suitable for a longer distance of bridging. Moreover, the serial link would be more cost efficient, due to the smaller number of wires, if it is utilized on a proper link on which the data traverses serially. Hence, the bridge scheme should support the serial bridge.
- *Flow Control:* In the on-chip interconnect the flow of data is controlled at two levels. 1) Link-level, which is performed locally at the link of *router-router* or *router-NI*, in order to control the flow at the data granularity of the flits. 2) End-to-End level, which is done globally at the end of the network communication channels i.e. NIs, in order to control the flow of NoC packets for the GT traffic [18]. We would see later in this section that it can affect the QoS provided by the bridge. The bridging scheme should provide both link-level and End-to-End flow control to every sub-NoC.
- *Buffering:* A bridge can be implemented with no buffer, one buffer, virtual-channel buffers, or virtual-circuit (per-connection) buffers. Each of these implementation options implies different features for the bridge. Although both the none-buffer and the one-buffer design have the lower implementation costs, the virtual channel and virtual circuit designs can distinguish among connections. Consequently, the virtual channel and virtual circuit designs can provide the proper QoS for every connections separately. This criterion is a trade-off between the QoS and cost requirements. Generally, the QoS has higher priority than the cost.
- *Path Manipulation:* This criterion indicates if bridging at a link would cause any changes in the determined path of a packet (e.g. number of travelling hops). There exists a time coupling between two ends of a link. If a link delay is too long that the data can not arrive to the other end at the required time, then there should be some intermediate hops added. Hence, the data packet headers are manipulated to compensate this longer path. This has a direct impact on the timing and implementation cost.
- *Frequency:* The frequency dependency of two ends of a link, located on different FPGA boards, is a challenging design issue. This is essential because of the TDM scheduling policy applied for GT traffic. Inserting the bridge on a NoC physical link at a proper protocol stack layer which has no frequency dependency between its ends, would make the design much simpler.
- *Separate Connections:* Similar to the buffering, in increasing QoS control, a bridge may not distinguish connections and treat all data equally, or distinguish different QoS classes (e.g. GT and BE), or distinguish data belonging to each connection.

In this work, we require a serial bridging solution that takes care of the both local and global flow control, and having a low cost. The bridge should preserve the connections' QoS, while solving the frequency coupling problem. For this purpose, we propose possible interconnect bridge implementations and compare them based on these criteria.

C. Protocol Stack Exploration

The main purpose of bridging is to split an interconnect into two (or more) smaller sub-networks. In this section we explore the possible cut-point of an interconnect to find the best place and scheme for bridging the sub-interconnects. Figure 1 illustrates a protocol stack model of an interconnect based on the proposal in [6]. The granularity of the data at each layer on every interconnect links is also shown.

Point 4 & 5 in the Figure 1, are *router-router/NI* links which can be selected at any link in the NoC.

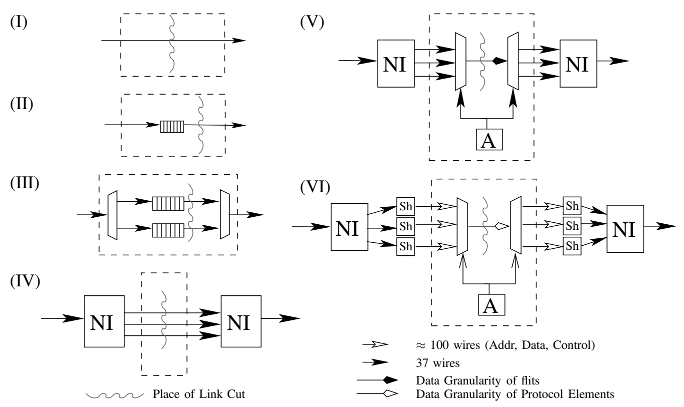

(I)
(II)
(III)
(IV)
(V)
(VI)
NI
NI
A
Sh
≈ 100 wires (Addr, Data, Control)
37 wires
Data Granularity of flits
Data Granularity of Protocol Elements
Place of Link Cut

Fig. 2. Bridging schemes based on the interconnect protocol stack

Physical Layer: According to the first bridge requirement (transparency) the lowest possible interconnect protocol stack layer is investigated. The first bridge proposal, which is scheme I, is implemented at the physical layer. Since it acts as the link wires, it deals the same with all traffic that might have different QoSes. There is a frequency dependency between two neighbor routers for the case of GT traffic. This scheme is too expensive in terms of matching frequencies and long delay in link-level flow control. So it is not a good bridging solution.

Link Layer: By adding a buffer to the previous implementation, resulting in Scheme II of Figure 2, we go up one layer in the protocol stack to the link level. In this case, while the bridge is similar to a pipelined NoC link which is transparent to the packets, it should take care of the link-level flow control. Here, the buffer is also shared among all traffic which might cause GT packets to be stuck behind BE packets. This might happen when a BE packet enters the buffer before the GT traffic and is waiting to be transfered by the bridge. Since the GT packets have higher priority they should not be waiting for BE ones to be transferred. Consequently, this scheme does not fulfill the bridge QoS requirement.

Network Layer: We would solve this problem by introducing another buffer in the scheme III. The buffers form two specific virtual channels (BE & GT). This makes the bridge to be at the Network level of the protocol stack, in which it performs as a *router*. It therefore manipulates the path as a new hop. The flow control at the link-level is performed for BE traffic, while the GT traffic flow is handled by the TDM scheduler.

Although the bridge QoS requirement problem is solved by Scheme III, both Scheme II-III still suffer from frequency dependency problem. The GT traffic requires not more than one (or at least integer number of) slot time delay between two neighbor hops. To achieve this for two ends of the bridge on different FPGA boards, the bridge design will be very complex. The complexity is to make TDM tables of the sub-NoCs coherent that the data is sent and received with the correct order [8]. Consequently, we propose Scheme IV in which a Network Interface is added at both ends of the bridge, shown in Figure 2, in order to remove the timing dependency imposed by TDM scheduling policy.

Transport Layer: Scheme IV not only raises the implementation level of the bridge up to the Transport layer of the protocol stack, but also it changes the bridge insertion point to the point 3 in Figure 1. The NI takes care of both link level and end-to-end flow control. It means that a connection is terminated in the first sub-NoC and on the other sub-NoC a new connection will be started. This technique solves the frequency dependency problem. Here, since the packets are de-multiplexed into the connections that they belong to, the QoS can be preserved per connection in this scheme. Although this scheme is very good in theory, there is a huge number of wires ($\simeq$ num_conn * 39) in parallel that the serial bridge would be very expensive to support them. Scheme V proposes an implementation of the bridge which includes connection arbitration and scheduling. This scheme fulfills the decoupling and QoS requirements of the bridge by a) having virtual circuit buffering per connection, b) terminating a connection at first sub-NoC to remove frequency dependency, c) forming a new packet at the second sub-NoC, and d) control link level and end-to-end flow of the data.

Session Layer: In scheme VI, the requirements are fulfilled the same as scheme V. This scheme corresponds to the points 1 and 2 in the Figure 1. However, the granularity of the data after and before the shell is different. So far in scheme I-V, data on a link has been a sequence of phits. A shell de-serializes this data to a parallel protocol specific data elements e.g., DTL *Command, Address, Data*. Hence, the bridge proposed in the scheme VI should either be implemented as a parallel bridge. Moreover, the interfacing protocol dependent data results in impossibility of interleaving the requests and responses of different connections at the fine grained elements e.g., Commands, Address, Data. This might cause the bridge to be blocked by a single transaction for a long time and deadlock may occur. The QoS support is therefore not possible.

TABLE I
EVALUATION OF THE BRIDGING SCHEMES AGAINST THE REQUIREMENTS.

Scheme	Transp.	Decoup.	QoS	Area Cost	Latency
I (Physical)	+	-	-	+	-
II (Link)	+	-	-	+	-
III (Network)	+	-	+	-	-
IV (Transport)	+	+	+	-	+
V (Transport)	+	+	+	+	+
VI (Session)	-	+	-	-	-

Finally, Table I summerizes the requirements comparison of the different bridging scheme implementations. A "+" means that a scheme fulfills a requirement, where a "-" means not. Since Scheme V fulfills all the requirements, it is the best bridging scheme. This is also a generic scheme that not only can be placed before and after an NI, but also it can be connected directly to a shell or a streaming port of an IP.

V. Bridge Architecture

In this section we present the detailed architecture of the bridge based on Scheme V illustrated in Figure 2. The bridge architecture block diagram is depicted in Figure 3.

Board-to-Board Interface: The proposed bridging scheme in this paper not only connects sub-NoCs which are embedded on different FPGA chips, but also provides the off-board

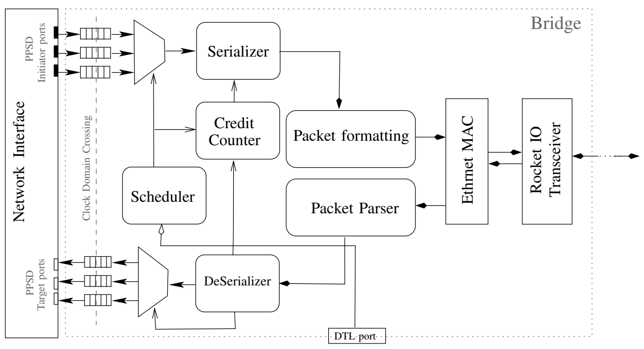


Fig. 3. The bridge architecture

interconnections. Typically the FPGA boards support common off-board interconnect protocols such as PCI, Ethernet, USB, and SPI. All these protocols transfer the data serially. USB and SPI are master-slave protocols which are not applicable to our proposal, whereas we might have both master and slave at both side of the bridge. PCI can not provide long distance wired connection. On the other hand, Ethernet provides a scalable long distance wired connection. In this paper we implement the Ethernet protocol by utilizing two hardware modules as shown in Figure 3. The physical layer of OSI protocol stack [5] is implemented by a Xilinx RocketIO transceiver; and the data link layer is realized by Medium Access Control (MAC) module. These modules are board and FPGA type specific.

Data Format: The Ethernet protocol implies the data transfer in a specific packet format which is illustrated in Figure 4. There are always 36 fixed bytes in the frame, while the payload data length is variable. The bridge inserts the data, to be transferred, in the payload field of the frame. Therefore, the longer the length of the payload, the more efficient link utilization. The bridge data consists of multiple connections data, and the credit values regarding the link-level flow control between two bridge modules of each sub-system. This is encoded in two least significant bits of each data byte as shown in Figure 4. Since the transmission of an Ethernet packet can not be stalled, the NULL byte is needed when there is no data in the buffers to be sent by the bridge. The NoC phits data related to a connection follows the credits value for that connection (encode as "10") and the connection identifier byte (encoded as "01"), respectively. In our NoC every phit is 37 bits which form 5 bytes in the payload.

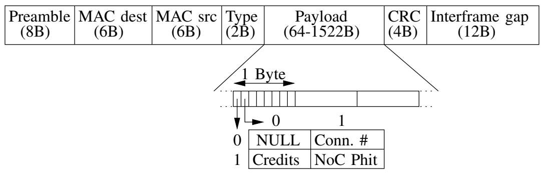


Fig. 4. Ethernet packet format with the bridged connections data encoded in the payload

Multi-Connections: Figure 3 shows that data of multiple connections data are buffered in the bridge at the first stage of transmitter side. A buffer is then arbitrated and its data transmission is scheduled. The data, which is at the granularity of phits, is serialized and transmitted in the Ethernet packet format. On the other side, the serialized data is converted again to the original phits and placed in its corresponding buffer. The reason of having a dedicated buffer for each connection, while there are connection buffers in the NI, is that the bridge generally can be also connected directly to a shell. Essentially, a bridge which is placed before a shell is to handle the link level flow control and to store the data to be accepted by the shell. Moreover, the frequency on which the bridge operates on is fixed to 125 MHz and is imposed by the Ethernet MAC and physical modules. This is different from the operation clock frequency of the NoC (50-200 MHz). The buffers therefore are dual-clock First-in-First-out (FIFO) buffers which also provide the clock domain crossing.

Scheduling and Arbitration: The arbitration and scheduling of the connections is realized as a TDM scheme. The TDM provides the QoS requirements by assigning time slots of the transmission link to every connection. The TDM connection table assignment is runtime programmable via a DTL port of the bridge in a memory-mapped access. The number of slots the TDM table that a certain connection has determines its throughput and latency over the bridge.

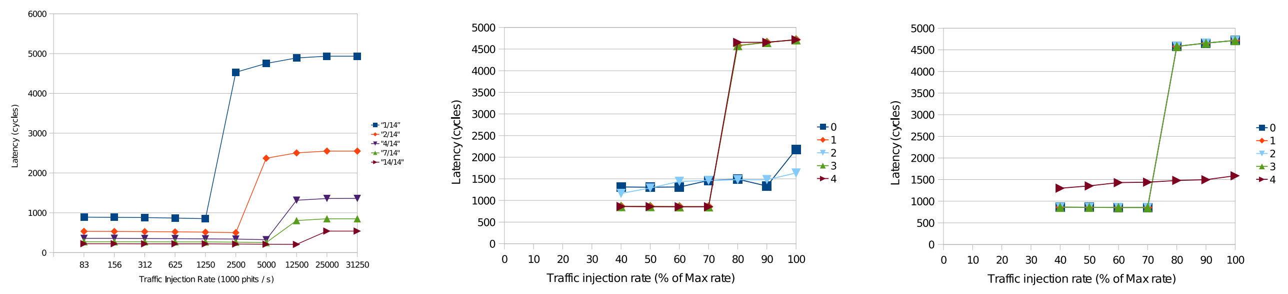
Flow control: It is mentioned in Section III that an NI takes care of the global end-to-end connection flow control and also the link level flit flow control to the neighbor *router*. Likewise, the NI does the same in a sub-NoC locally for the bridged connections and links. However, there is also a global flow control of data over the bridge needed. This flow control is performed using a credit-based technique. The transmitter side is initialized with the initial free space of each corresponding buffer at the receiver side and is decremented during the data transmission. As soon as a credit payload is received, the credit counter is updated by adding the new value. The credit transmission from receiver side is guaranteed by the TDM scheduler when in every slot first the credit of corresponding connection is sent. Hence, the bridge is deadlock free.

VI. Experiments

A. Standalone Bridge

The bridge is synthesized for Xilinx Virtex 5 and 6 to be implemented on ML510 and ML605 emulation boards respectively. The embedded tri-mode Ethernet MAC Wrapper, specifically designed for Virtex 5 and 6 FPGAs by Xilinx, is used as the MAC layer module of the bridge. The physical Ethernet interface is also available off-chip on the ML510 and ML605 boards. The bridge area cost is less than %1 of the FPGA resources, excluding the Ethernet MAC and Physical layer modules. The bridge is synthesized to handle 4 connections with a TDM table of 16 slots and 64-phit (64*37 bits) buffer sizes.

Figure 5(a) shows the network (NI-NI) latency versus the throughput of the bridge for different number of assigned slots to a certain connection in the TDM table of the scheduler, assuming the credits are available. The initial flat region in the graphs shows that connections do not use their entire allocated guaranteed bandwidth. Increasing the offered load beyond the guaranteed bandwidth budget increases the NI-NI latency to a finite maximum, since the waiting time outside



(a) (b) (c)

Fig. 5. Latency vs. throughput for one connection with variable slots assigned (a), for the connections initiated in the example platform from left sub-system (b) and from right sub-system (c).

the NI (between IP and NI) is not included here. Allocating more slots increases the bandwidth and lowers the latency.

B. System Level

Considering the set-up example platform in Figure 6, there are five connections across by the bridge. The connections traffic is initiated at the left sub-system by Tile0 and at the right sub-system by Tile1. The traffic QoSs are also shown in the figure. Figure 5(b) shows the latency versus throughput for the traffic initiated from the left sub-system. The throughput is variable according to the percent of the maximum possible ideal bandwidth provided by the bridge based on the number of slots assigned to a connection in the TDM table. The connections 0 and 2 have the lowest latencies for the higher bandwidths, since they are GT traffic. Connections 3 and 4 are the credits of the corresponding connections initiated from the right sub-system. Their data traffic is shown in Figure 5(c). The lowest latency belongs to the GT traffic of connection 4. The data traffic 0, 1 and 2 are the credit value of the corresponding connections initiated from the left sub-system. The graphs which are illustrated in these figures, follow the similar pattern of the graphs shown in Figure 5(a). Therefore, the bridge preserves the QoS of the connections while transferring the traffic to the other sub-system.

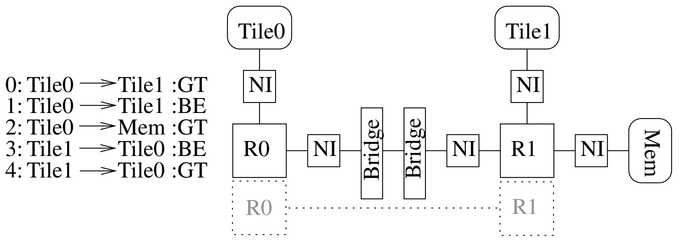


Fig. 6. The example platform setup

VII. Conclusions

In this paper we have proposed a generic on-chip interconnects bridging scheme between sub-SoCs implemented on different FPGA boards. For this purpose the protocol stack layers of the on-chip interconnect are investigated to realize the best level of bridge implementation at possible interconnect links. The outcome is a bridging scheme at the transport layer which fulfills the bridging requirements in terms of transparency, decoupling, cost and application quality of service preservation. We have evaluated the bridge implementation under variable traffic loads and the results showed that the bridge preserves the traffic quality of services. The area cost of the bridge is less than %1 of the FPGA resources. The SoC partitioning technique to generate the bridged sub-systems is left as an open future research work.

References

[1] W. J. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," in *Proc. Design Automation Conference (DAC)*, 2001.

[2] F. Steenhof, H. Duque, B. Nilsson, K. Goossens, and R. P. Llopis, "Networks on chips for high-end consumer-electronics tv system architectures," in *Proceedings of the conference on Design, automation and test in Europe (DATE)*, 2006, pp. 148–153.

[3] C. Chang, J. Wawrzynek, and R. W. Brodersen, "BEE2: A high-end reconfigurable computing system," *IEEE Design and Test of Computers*, vol. 22, pp. 114–125, 2005.

[4] A. Hansson and K. Goossens, "Trade-offs in the configuration of a network on chip for multiple use-cases," in *Proc. Int'l Symposium on Networks on Chip (NOCS)*, May 2007, pp. 233–242.

[5] J. D. Day and H. Zimmerman, "The OSI reference model," in *Proc. of the IEEE*, vol. 71, 1983, pp. 1334–1340.

[6] A. Hansson and K. Goossens, "An on-chip interconnect and protocol stack for multiple communication paradigms and programming models," in *Int'l Conf. on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, Oct. 2009.

[7] E. Beigne and P. Vivet, "Design of on-chip and off-chip interfaces for a GALS NoC architecture," in *12th IEEE International Symposium on Asynchronous Circuits and Systems, 2006.* , March 2006.

[8] S. Evain, J.-P. Diguet, and D. Houzet, "NoC design flow for TDMA and QoS management in a GALS context," *EURASIP J. Embedded Syst.*, vol. 2006, no. 1, pp. 4–4.

[9] X. Li and O. Hammami, "Multi-FPGA emulation of a 48-cores multiprocessor with NoC," in *3rd Int'l Design and Test Workshop (IDT)*, Dec. 2008, pp. 205–208.

[10] A.-M. Kouadri-Mostefaoui, B. Senouci, and F. Petrot, "Scalable multi-FPGA platform for networks-on-chip emulation," in *Application - specific Systems, Architectures and Processors (ASAP), 2007*, July 2007.

[11] Kouadri-Mostefaoui, Abdellah-Medjadji, B. Senouci, and F. Petrot, "Large scale on-chip networks : An accurate multi-FPGA emulation platform," in *11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools (DSD)*, Sept. 2008.

[12] B. P. Kommineni, R. Srinivasan, R. Holsmark, A. Johansson, and S. Kumar, "Modeling and evaluation of a NoC-internet interface," in *Swedish System on Chip Conference, Bstad, April 2004*.

[13] Y. Yin and S. Chen, "Design and implementation of a inter-chip bridge in a multi-core SoC," in *Int'l Conf. on Design & Technology of Integrated Systems in Nanoscal Era (DTIS), 2009*, April 2009.

[14] A. Patel, C. A. Madill, M. Saldana, C. Comis, R. Pomes, and P. Chow, "A scalable FPGA-based multiprocessor," in *Proc. of the 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2006.

[15] *Device Transaction Level (DTL) Protocol Specification. Version 2.2*, Philips Semiconductors, Jul. 2002.

[16] *AMBA AXI Protocol Specification*, ARM, Jun. 2003.

[17] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch, "Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip," in *Proc. DATE*, 2004.

[18] K. Goossens and A. Hansson, "The Aethereal network on chip after ten years: Goals, evolution, lessons, and future," in *Proc. Design Automation Conference (DAC)*, Jun. 2010.

[19] D. Wingard, "Socket-based design using decoupled interconnects," in *Interconnect-Centric Design for Advanced SoC and NoC*, J. Nurmi, H. Tenhunen, J. Isoaho, and A. Jantsch, Eds. Kluwer, 2004, ch. 15, pp. 367–396.