Parallel Computing 39 (2013) 424–441

Contents lists available at SciVerse ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

A hardware/software platform for QoS bridging over multi-chip NoC-based systems

Ashkan Beyranvand Nejad [a,*], Anca Molnos [a], Matias Escudero Martinez [a], Kees Goossens [b]

[a] Computer Engineering Lab., Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands
[b] Electronic Systems Group, Eindhoven University of Technology, Den Dolech 2, 5612 AZ Eindhoven, The Netherlands

ARTICLE INFO

Article history:
Available online 28 April 2013

Keywords:
Distributed SoCs
Bridge architecture
Networks-on-Chip
Quality of service
Prototype
Verification

ABSTRACT

Recent embedded systems integrate a growing number of intellectual property cores into increasingly large designs. Implementation, prototyping, and verification of such large systems has become very challenging. One of the reasons is that chips/FPGAs resources are limited and therefore it is not always possible to implement the whole design in the traditional system-on-a-chip solutions. The state-of-the-art is to partition such systems into smaller sub-systems to implement each on a separate chip. Consequently, it requires interconnecting separate chips/FPGAs. Since Networks-on-Chip (NoCs) have become common interconnection solutions in embedded designs, we propose to bridge NoC-based SoCs enabling a generic multi-chip systems interconnection. In this context, the contribution of this paper is threefold, (i) we explore the NoC protocol stack to determine the best layer for implementing the off-chip bridge, (ii) we propose a generic hardware architecture for the bridge, and (iii) we develop a new software architecture enabling seamless configuration and communication of multi-chip NoC-based SoCs. Finally, we demonstrate performance, i.e., bandwidth and latency, of the bridge in a multi-FPGA platform, while the bridge guarantees QoS of traffic. The synthesis results indicate the implementation area cost of the bridge is only 1% of Xilinx Virtex6 FPGA.

1. Introduction

The demand of executing more and more applications on electronic consumer devices has led to an increase in the size and the complexity of recent embedded systems. On one hand, modular design approaches have enabled integration of numerous processor cores, memories and peripheral IPs into an embedded system; on the other hand, implementation, prototyping and verification of such large designs on chips/FPGAs have become more complex and challenging as follows.

Implementation: although Moore's law indicates that the number of transistors that can be placed on a chip doubles approximately every two years, the recent designs can be so large that a single chip's resources are still limited. Recently, there have been some efforts to bring 3D IC stacking on a single chip into practice [1], however, traditionally, system designers partition such large systems into smaller sub-systems to implement them as several Systems-on-Chip (SoCs), or accompany SoCs with a number of *companion* chips [2]. In such systems, denoted as multi-chip systems, each SoC has to communicate with other SoCs and therefore a multi-chip interconnection scheme is essential.

Prototyping: FPGA prototyping has become a common approach for early software development and hardware design verification [3]. The limited resources of an FPGA, however, are not sufficient for prototyping the current large SoCs. A system is therefore required to be partitioned into number of sub-systems, each of them is implemented on a single FPGA chip [4], and

* Corresponding author. Tel.: +31 (0)15 27 83644; fax: +31 (0)15 2784898.
E-mail addresses: A.BeyranvandNejad@tudelft.nl (A.B. Nejad), A.M.Molnos@tudelft.nl (A. Molnos), K.G.W.Goossens@tue.nl (K. Goossens).

0167-8191/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.parco.2013.04.011

therefore a multi-FPGA system is formed [5]. Potentially, the FPGA chips are located on different circuit boards which are physically decoupled. Such systems require multi-chip interconnection mechanisms that also support board-to-board multi-FPGA communications.

Verification: both the implementations and prototypes of the current complex SoCs may still contain errors. Debug and verification processes are essential to find the erroneous parts of the systems and verify the systems' correct functionality. Therefore, an external, fast, non-intrusive access to on-chip systems is required for two purposes: (i) to retrieve trace of on-chip data off-chip, and (ii) to enable an external *host* to perform debug and verification actions [6].

The common problem in the aforementioned trends is a need for an off-chip communication scheme for interconnecting individual SoCs. For this purpose, several techniques have been proposed in the literature, such as the work in [7–9], however to the best of our knowledge none of them provides a generic solution to answer the need of inter-chip, inter-FPGA, and chip/FPGA-host communications. In this work, we propose a generic, efficient hardware and software architecture for off-chip interconnection of individual SoCs. The SoCs internally have their own communication mechanisms to interconnect the cores and IPs. The off-chip interconnection mechanism is efficient when it is compatible with and seamlessly integrates in the on-chip interconnections. Since recently Network-on-Chip (NoC) [10] has become a common on-chip interconnection technology, our proposed off-chip interconnection scheme is adapted to be compatible with NoCs' properties.

Various NoC architectures exist in the literature, e.g., Æthereal [11], Nostrum [12], Mango [13], QNoC [14], that may offer one or more quality of service classes. Many applications, e.g., signal processing and video streaming, have timing and throughput demands each require a specific QoS such as Guaranteed Throughput (GT) or Best Effort (BE). Consequently, the off-chip communication scheme should be also able to offer different QoS classes for the traffic of interconnected sub-systems to be compatible with the target on-chip interconnects.

1.1. Contributions

In this paper, we propose a generic, efficient technique for interconnecting a NoC-based system with other (NoC-based) SoCs or any other external IP. In the rest of this paper, we refer to this scheme as *bridging*, since it makes a bridge for traffic from/to a chip to/from another chip/IP.

To make the bridge fully compatible with the on-chip interconnects we establish four design requirements of the bridge as follows; (i) the bridge should seamlessly extend the NoC such that memory-mapped accesses remain unchanged from applications point of view. In other words, in the global memory space of the system the bridge is *transparent* such that the memory-consistency model of the system is preserved, (ii) multi-chip bridging at the circuit board level should be supported, and the bridge should *decouple* temporally and physically the systems implemented on different circuit boards, (iii) the *quality of service* offered by the overall interconnect (i.e., sub-NoCs + bridge (s)) to the applications should be preserved, and (iv) the bridge should be implemented efficiently with high *performance* in terms of bandwidth and latency, and with low *area cost*. In the context of a bridging scheme that fulfills all these requirements, the contribution of our paper is threefold, as follows.

First, we investigate a NoC-based system to identify the possible bridge insertion points (i.e., links). At each link, we study different layers of the on-chip interconnect protocol stack [15] for possible bridging schemes. Our protocol stack model is based on the proposal in [16], where the model consists of five layers referred to as *session*, *transport*, *network*, *data link*, and *physical* layer. Our design space investigation is driven by the NoC properties that have impacts on the bridge's requirements. We refer to these properties as *design options*, and we identify them as following: (i) parallel/serial link, (ii) flow control, (iii) buffering, (iv) routing, (v) synchronicity, and (vi) QoS. The investigation results in a novel proposal for a bridge design at the transport layer of the NoC.

Second, we propose a hardware architecture for the bridging scheme. The architecture is generic in the sense that it supports all inter-chip, inter-FPGA, and chip/FPGA-IP systems. We implement an HDL version of the bridge kernel that supports generic number of NoC connections each of which may be either best-effort or guaranteed-throughput.

The third contribution of this work is a software architecture to configure and to enable transparent global memory space communications over the bridged sub-systems. This architecture extends the run-time NoC configuration technique proposed in [17] and enables the bridge to integrate with and play as an on-chip host for NoC-based SoCs.

For our experiments, we set up a multi-FPGA system in which two instances of the bridge is utilized; one to connect two sub-NoCs each implemented on a separate FPGA, and another bridge to connect to an external host, which is a PC in our case. The experimental results show that the bridge achieves high-performance in terms of bandwidth and latency, and justify that it is able to provide required QoSs to data traffic.

1.2. Organization

The rest of this paper is organized as follows. In Section 2 we review the related work. Section 3 gives an overview of our target on-chip interconnection network. In Section 4, the bridge design requirements and the design options are explained in details. Based on them, in Section 5, we investigate the NoC's links and the protocol stack to propose the best bridging scheme. The bridge hardware architecture is proposed in Section 6 followed by the proposal for the software architecture in Section 7. The experimental results of stand-alone bridge and of the case where the bridge is used in a multi-FPGA set-up are presented in Section 8. Finally, Section 9 concludes the contributions of this paper.

2. Related work

There are three main research topics that are directly related to our work. This section discusses them in turn, as follows. First, the existing techniques in on-chip and off-chip signaling is introduced to compare the related work that targets NoC-based systems with our proposed technique. Second, the related work in on-chip interconnect protocol stack is discussed; and third, we review interconnect configuration techniques.

Traditionally, a large system is built and prototyped by using multi-chip/FPGA technology where a number of single packaged ICs communicates together [2]. The communication between chips is referred to as off-chip signaling when the chips are totally separated and may or may not be placed on a common circuit board. To increase the system performance and density, a Multi-Chip Module (MCM) [18] (denoted as FPMCM in FPGA-based system [19,20]) is proposed as an advanced technology in integrating multiple chips in a package. It is also known as System-In-Package (SIP). The communication between chips in MCM is realized by off-chip signaling. Recently, 3D-IC technology is introduced to integrate multiple IC layers both vertically and horizontally on a single circuit chip [1]. The communication in this technology is on-chip signaling based, since the communication between layers is realized by new techniques such as Through-silicon via [21] and offer on-chip signaling communication quality.

In on-chip signaling techniques, a system is implemented on a single chip and the interconnect is at the level of signals' (wires) place and route on one chip. However, in off-chip signaling, interconnects are at the system level of multiple chips. Usually in these cases, the unified system has some requirements to be met in executing applications. According to these signaling definitions our bridging scheme is positioned as off-chip signaling communication technique in multi-chip systems. In spite of this, our bridge is also potentially applicable to on-chip signaling systems when they require seamless, guaranteed QoS off-chip interconnection.

Significant research efforts have been done in different aspects of interconnecting multiple chips or FPGAs. These aspects regard either the bridging over the chips that are located on the same circuit board (on-board bridging) or bridging over different boards (off-board bridging). Prior work proposed network technologies that could offer both on-chip and off-chip communications [7–9]. The work in [9] presents on-chip and off-chip networks for modeling large-scale neural systems. The off-chip network is to offer a multi-chip interconnection at the network level. It implements a handshake-based protocol on the parallel link. The authors of [7] propose a photonic NoC, in which optical fibers enable the seamless extension of interconnects over multiple chips. Furthermore, an off-chip NoC interface is proposed in [22], where the interface is implemented at the physical layer in order to offer a transparent NoC protocol for different NoC-based subsystems. The interface has a parallel connection of 78 signals between chips per each bidirectional link, which is expensive and therefore it is not applicable to off-chip bridging of the chips located on different circuit boards.

The concept of interconnecting sub-NoCs is introduced in [23]. The sub-NoC interconnection is realized at the network layer by a synchronizer module. The proposed technique in this work can preserve the QoS of connections over sub-NoCs which are implemented on the chips located on one circuit board. A 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 quality of traffic. We do not need such an adaptation in our design since the bridge fully decouples the sub-NoCs that may also be implemented on the chips located on different circuit boards.

A hierarchical NoC architecture is introduced in [8], to support multi-chip platforms. The target technology of this work is multi-FPGA multimedia systems. The NoC has a *gateway* module beside the typical basic building blocks, i.e., routers and network interfaces. The gateway is responsible to translate the addresses between two levels of the NoC hierarchy. Therefore, multi-chip bridging is only feasible between the different levels of hierarchy and requires address translation. Compared to this solution, our bridge maintains the global memory map model while no address translation between bridged sub-systems is needed.

The authors of [24,3,25,26] introduce different multi-FPGA platforms for emulating large ASIC designs. The work presented in [27] shows how a multi-processor SoC (MPSoC) with 48 cores can be fitted in 4-FPGA platform by extending a NoC with off-chip synchronous links. A multipurpose emulation platform which is used with different NoC topologies is also presented in [4,28]. Network links between routers, which are placed in different chips, are emulated by using high speed serial links as both inter-chip and inter-board connections. However, this proposal does not provide the QoS management for the connections traffic.

A bridge design that acts as a network interface is implemented in [29]. This bridge operates at the transport layer of the NoC and at the application layer at the Internet side. The work presented in [30,31] present multi-processor systems that can take advantage of the intrinsic multi-hop nature of the NoC to allow transparent inter-chip connection to IPs. In [30], 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 layer. 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 buffer. Comparing to our proposal, we implement the bridge at the transport layer such that the frequency dependency between two ends of the bridge is not an issue.

It is previously mentioned in Section 1 that the bridging scheme should be compatible with the on-chip interconnect. Therefore, a well-structured overview for the on-chip interconnect protocol stack is essential. Among the many protocol

stacks for NoCs proposed in the literature [32–36,16], we have found the work in [16] to be an accurate and complete proposal that applies to our NoC. Since one of the requirements of the bridge is to seamlessly extend the NoC, the global memory map model of the overall system should be maintained. The protocol stack proposal in [16] considers both streaming and distributed shared memory communication, and it is therefore enable designing a bridge to be compatible with the NoC at these layers.

Most of nowadays embedded systems require run-time (re) configuration technologies [37,17]. Usually the host, which is responsible for (re) configuration, is on-chip. However, when a system is partitioned to be implemented on multiple chips or FPGAs the host would be placed on the same chip of one of the sub-systems, and therefore it becomes remote for the other ones. Our bridging scheme can be used in any configuration scheme that requires an external remote access by a host to on-chip interconnects. A (re) configuration technique of NoC-based systems is proposed in [17]. This technique uses the functional connections of the NoC to (re) configure the local system by accessing memory-mapped programmable locations. Here, we extend this technique to configure remote sub-systems that might be off-chip. Our experiments demonstrate the overhead of the remote configuration and compares it with the proposed solution in [17].

To the best of our knowledge, none of the existing work has investigated all the possible interface layers of bridging to find a scheme that could satisfy efficiently both on-board, either on-chip or off-chip, and off-board multi-chip interconnection, while respecting the application requirements. All the approaches 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 implements both off-board and on-board multi-chip bridging. The bridge manages the QoS of applications, while it is connecting the SoCs implemented on the different circuit boards.

3. On-chip interconnect overview

The on-chip interconnection network has a key role in the bridging scheme. In this section we give an overview of the interconnection network. 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 Fig. 1. The on-chip interconnect consists of traditional *bus* technology and a NoC architecture. The *master* starts a request by sending a *write* or *read* command to the bus (point 1). 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., OCP [38], DTL [39], AXI [40]). A shell serializes/de-serializes the protocol specific data elements (*commands*, *address*, *data*) to/from a Point-to-Point Streaming Data (PPSD) [16] in a handshake-based interfacing protocol at point 3. The PPSD is then packetized/de-packetized 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 one clock cycle on a *hop* of the interconnect. The packets reside in NI buffers waiting to be sent out. The flits are directed by *routers* to the links (point 5) on a path determined by the flit header of a packet [11], or by a routing table inside the router. Once the request reaches the destination NI, the above procedure occurs in reverse until the slave accepts the request. The bus at a slave side enables the connection of multiple memory-mapped masters to the slave. The slave's response follows the same scenario to reach at the master side.

A connection is formed by a request channel and a response channel. The interconnect may treat connections differently based on their service classes. The service provided for a connection data may be 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 [12] and Æthereal [11], but it can be equally applied to any other NoC topology.

4. Bridging requirements and design options

The requirements for a multi-chip bridge have direct impact on the bridging scheme. Therefore, in this section we first establish the design requirements as follows.

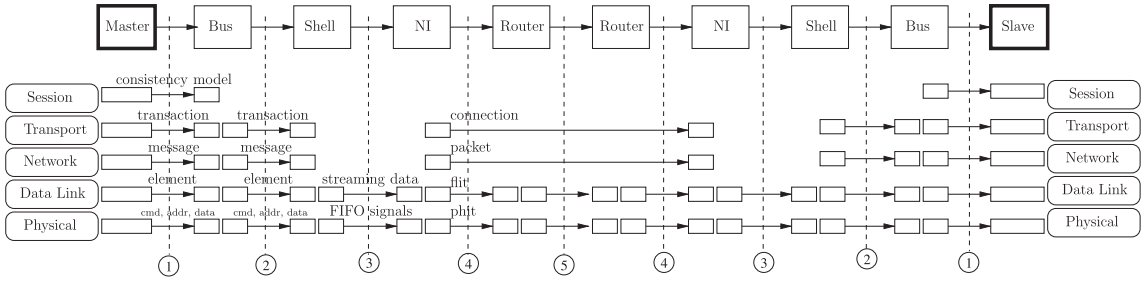

Fig. 1. Overview of on-chip interconnect connection with the involved protocol stack layers at the links.

(1) *Transparency*: ideally, from an application point of view the bridge should be invisible, i.e., the global memory consistency model should be maintained. This is especially essential in the case of FPGA emulation of a partitioned system, where the emulated system should be functionally as close as possible to the real prototype. In this case, from an application's point of view, the overall interconnect should behave the same as in the original system, and therefore no manipulation in the path of the packets traveling over the bridge is acceptable.
(2) *Decoupling*: two sub-systems could be implemented on different chips physically located on separate circuit boards. The bridge therefore should support the full decoupling between the two systems [41]. Decoupling includes isolation of both physical and temporal dependencies, e.g., voltage levels and frequency.
The physical dependency that is due to the board-to-board bridging, would be solved by the off-board interfacing protocol of the bridge. Temporal dependency comes from the fact that the NoC routing algorithm may assume a relative timing between consecutive network hops. This means that each hop expects to receive the flits in a specific moment in time. For instance, consider a NoC that uses circuit-switching routing method. A Time Division Multiplexing Access (TDMA) scheduler schedules flits according to its pre-programmed time-slot table in NIs [23]. This implies a timing dependency between interconnect hops, in which the flits should arrive in the next hop at a specific time according to the TDMA table. The bridge should either resolve or work around the temporal dependency between two hops.
(3) *Quality of service (QoS)*: the bridge should preserve the traffic QoS that is provided by the NoC to the applications, e.g., no GT traffic should wait for any BE traffic to be serviced. If there is only one QoS class supported by the NoC topology, all connections are treated equally.
(4) *Area and performance*: since the bridge is meant to be implemented in silicon or in an FPGA, low area cost is desirable. The bridge area cost is the number of transistors that are utilized to build logic elements, buffers and memory blocks, plus the number of pins used for the off-chip link. In general, buffers and memory blocks are expensive resources in both silicon and FPGA, where usually the pins are scarce resources.

The bridge performance is the throughput and the latency, as for any communication medium. Since both transparency and QoS requirements aim to lower the possible impact of the bridge on the traffic, the bridge should provide the traffic's required throughput. To a large extent, the bridge's throughput depends on the off-chip communication link. However, since the bridge supports multiple connections and provides proper QoS to the traffic, the bridge's bandwidth is shared between a number of NoC connections. This sharing results in a limited throughput for each connection. The latency overhead of the bridge is inevitable, but we aim to lower this as much as possible by designing the bridge in such a way that the end-to-end latency is minimized.

In order to fulfill four aforementioned bridging requirements, the bridge design options should be considered carefully. The design options are selected properties of the NoC that can have either direct or indirect impact on designing the bridge architecture, as follows.

- *Parallel/Serial Link*: An interconnect physical link can be either parallel or serial. Accordingly, a bridge might be implemented as a serial or a parallel interconnect on the link. The parallel implementation is faster, and it is suitable for a short distance of bridging. The serial one is slower and is suitable for a longer distance bridging because a few number of wires should be routed. Since the pins of a chip are scarce resources, the serial link would be more cost efficient as it requires the smaller number of wires and interfacing pins. Hence, the bridging scheme should preferably support the serial link.
- *Flow Control*: In the on-chip interconnect the flow of data can be controlled at two levels, (i) at link-level, where it is performed locally at the links between *router-router* or *router-NI*, in order to control the flow at the data granularity of the flits, and (ii) at end-to-end level. At this level the flow of NoC packets is controlled for the GT traffic globally between two ends of the network communication channels, i.e., NIs [11]. The level of the flow control that can be supported by the bridge, affects the traffic QoS. The bridging scheme should hence provide both link-level and end-to-end flow control to the sub-NoCs at each side, and implement a flow control mechanism for off-chip links.
- *Buffering*: Buffering is an important technique in the NoC to implement different routing algorithms and flow control mechanisms. The routers may have separate buffers for each class of QoS, and NIs have number of buffers per connections. A bridge might be implemented with no buffer, one buffer, virtual-channel buffers, or virtual-circuit (per-connection) buffers [42]. Each of these implementation options implies different features for the bridge that we discuss later in Section 5. Although the none-buffer and the one-buffer design have the lower implementation costs, the virtual channel and virtual circuit designs can distinguish among connections, and consequently, such designs can provide the proper QoS for every connection separately. This option is a trade-off between the QoS and cost requirements. Generally, the QoS has higher priority than the cost.
- *Routing*: In the NoC, the path manipulation only occurs in the routers. Therefore, routing options indicate if bridging at a link would cause any changes in the determined path of a packet (e.g., number of traversed hops). For example, if temporal coupling may exist between two ends of a NoC link, i.e., router-router or NI-router, and the link delay becomes long due to the bridge insertion such that the data cannot arrive to the other end at the required time, there should be some intermediate hops added to transfer data in more than one step. Hence, the data packet headers should be manipulated to include the added hops. This has a direct impact on the transparency, timing, and implementation cost. A transparent bridge should not alter any routing path of the packets, and therefore it should not act as a router itself.

- *Synchronicity*: In a synchronous NoC architecture, a frequency dependency exists between two ends of a link. This dependency is tighter, if the routing algorithm applies also more timing constraints such as circuit-switching method by TDMA flits scheduling as explained previously in the decoupling requirement of the bridge. If in a partitioned system, the two ends of a link are located on different chips in different circuit boards, the design of a bridge would be very challenging because the system frequency at one side can be very different and out of sync from the other side. Inserting the bridge on a NoC link at a proper protocol stack layer which relaxes the frequency dependency between the two ends, would make the design much simpler.
- *Quality of service*: The NIs distinguish among different connections traffic to provide proper routing services to each class of QoS. With respect to QoS control, a bridge may either buffer all different connections' traffic in one queue, or buffer the traffic separately according to the QoS classes (e.g., GT and BE). The latter is one of the bridge requirements; therefore, the bridge should be implemented on a link that makes this possible.

To summarize, the discussion of design options suggests that the bridging scheme should be a serial implementation that takes care of the both local and global flow control. The bridge could preserve the connections' QoS if it would implement proper buffering method, and the bridge insertion on a proper NoC's link solves the frequency coupling problem. In the following section, we propose possible bridge implementations and compare them based on these options.

5. Protocol stack exploration

A connection between a *master* and a *slave* is formed by a set of physical communication links through the network, e.g., *router–router* and *router-NI* link. The links are illustrated as numbered points in Fig. 1. Possibly the bridge could be instantiated at each of these links. In this way, the bridge either partitions a NoC into two sub-networks, or connects two different NoC-based systems at these links.

Moreover, there is more than one layer of the interconnect protocol stack which is involved at a communication link. This implies that the bridge can be also implemented at different layers of the protocol stack. In this section, we explore the interconnect protocol stack layers in every interconnect links. We present a bridging scheme at each layer and we discuss the schemes based on the bridging requirements to realize the most proper solution that fulfills all the requirements. Fig. 1 illustrates a protocol stack model of an interconnect based on the proposal in [16]. The granularity of the data at each layer on every interconnect links is also shown. Accordingly, the links are numbered in the figure and we study the possible bridging schemes by discussing the characteristics of the bridge at each of these points, in turn.

5.1. Physical layer

Every *router-NI* and *router-router* link of the NoC is represented by Point 4 and 5 in Fig. 1. Ideally, the bridge is required to be transparent. This means that the bridge behaves like a wire at physical layer, if it is implemented at these points. Scheme I in Fig. 2 illustrates such a bridge. The bridge therefore deals equally with all traffic that might have different classes of QoS. Moreover, if in a NoC technology there is a frequency dependency between two neighbor routers, this scheme would be

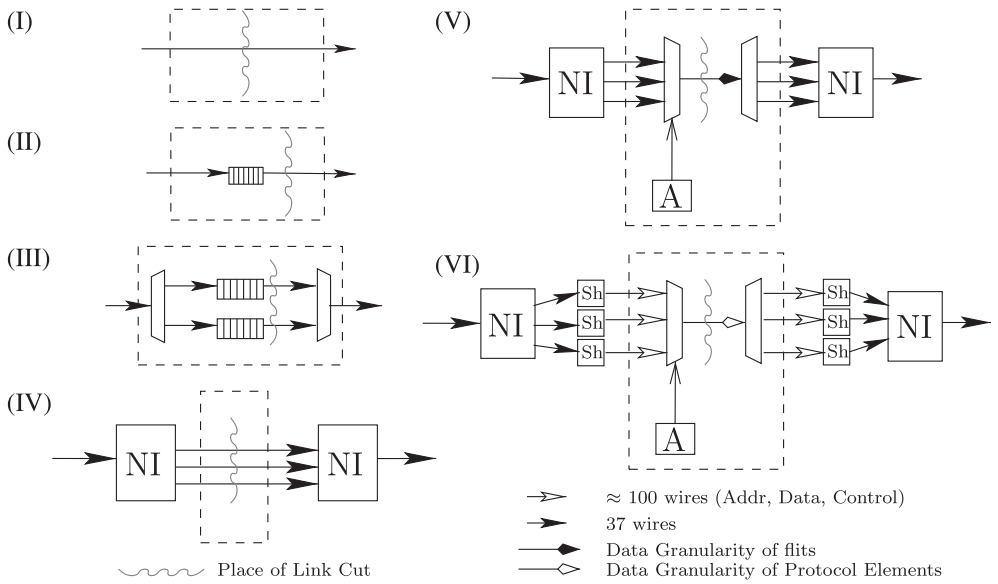

Fig. 2. Bridging schemes based on the interconnect protocol stack.

expensive because of frequencies have to be matched between two systems implemented on different circuit boards, as discussed in Section 4. Beside this, link-level flow control has a large delay due to the long link between the two ends of the bridge.

5.2. Data-link layer

To remove the long delay in the link-level flow control, this link could be pipelined. This means that a buffer could be added to the previous bridge implementation. In this case, the link-level flow control is done between the buffer and each end of the link. It results in Scheme II of Fig. 2, which is elevated one layer up in the protocol stack to the data link level, but it is still on Point 4 and 5 of the interconnect. Since the bridge is similar to a pipeline link, it is transparent to the packets. while the buffer is shared among all traffic, GT packets may be stuck behind BE packets when a BE one enters the buffer before the GT one and is waiting to be transferred 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.

5.3. Network layer

To separate traffic with different QoSs, we add another buffer in parallel with the existing one of Scheme III. The buffers form two dedicated virtual channels for BE and GT class of QoS. This bridge is therefore implemented at network layer and still on Point 4 & 5. At this layer, the bridge performs as a *router* as it parses the packets' header to distinguish between BE and GT ones. It therefore manipulates the traffic path while it appears as a new hop, and consequently the NoC topology is changed. The flow is controlled for BE traffic at the link-level, while the GT traffic flow is handled end-to-end by NIs.

All the schemes that we have discussed so far, including the network layer one, have frequency dependency between two ends of the bridge. For example, in case of TDMA circuit-switching routing, the GT traffic requires not more than one (or at least an integer number of) time slot delay between two neighbor hops. Fulfilling this constraint on the link delay becomes more complicated when the bridge is implemented between two chips that might be located on different boards. The main challenge is to make TDMA tables of the sub-NoCs coherent such that the data is sent and received in the correct order [23]. To work around this problem, we propose Scheme IV in which a network interface is added at both ends of the bridge as shown in Fig. 2. Now, we discuss the advantages and disadvantages of this scheme as follows.

5.4. Transport layer

Scheme IV not only raises the implementation level of the bridge up to the transport layer of the protocol stack, but it also changes the bridge insertion point from Point 4 & 5 to Point 3 in Fig. 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 a new connection is started on the other sub-NoC. This scheme solves the frequency dependency problem due to the fact that once the packets are arrived in the NI, there is no timing dependency afterward. Here, since the packets are de-multiplexed into their dedicated connection queues, the QoS can be preserved per connection. Although this scheme sounds promising in theory, in existing NoC implementations, there are a large number of wires in parallel (e.g., 39 wires per connection in Æthereal) and this leads to a large cost.

Scheme V proposes an implementation of the bridge which arbitrates between connections. In this scheme, the bridge schedules one connection's data transfer at a time to reduce the number of wires. This scheme also fulfills the decoupling and QoS requirements of the bridge by (i) having virtual circuit buffering per connection, (ii) terminating a connection at first sub-NoC to remove frequency dependency, (iii) forming a new packet at the second sub-NoC, and (iv) control link level and end-to-end flow of the data. Hence, this scheme is a promising solution for our bridging requirements.

5.5. Session layer

In Scheme VI, the requirements are fulfilled the same as Scheme V. This scheme corresponds to Point 1 & 2 in Fig. 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 the data to parallel protocol specific elements, e.g., *command, address and data* in DTL, and therefore the bridge in this scheme should be implemented as a parallel bridge. Moreover, when the data is dependent on the interfacing protocol elements, interleaving the requests and responses of different connections at the fine

Table 1
Evaluation of the bridging schemes against the requirements.

Scheme	Point in Fig. 1	Transparency	Decoupling	QoS	Area cost	Performance
I (Physical)	4,5	+	–	–	+	–
II (Data-Link)	4,5	+	–	–	+	–
III (Network)	4,5	+	–	+	–	–
IV (Transport)	3	+	+	+	–	+
V (Transport)	3	+	+	+	+	+
VI (Session)	1,2	–	+	–	–	–

grained elements (e.g., *commands*, *address and data*) becomes impossible. Such interleaving might cause the bridge to be blocked by a single transaction for a long time and a deadlock may occur. Consequently, GT traffic may be blocked by a BE one that invalidates the QoS support.

Finally, Table 1 compares the different bridging schemes. A "+" means that a scheme fulfills a requirement, whereas a "−" indicates otherwise. Since Scheme V fulfills all the requirements, hence it is the most suitable bridging scheme for our requirements. 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.

6. Bridge hardware architecture

In this section we present the detailed architecture of the bridge to implement Scheme V illustrated in Fig. 2. The bridge architecture diagram is depicted in Fig. 3(a). The bridge Kernel consists of five main units that are responsible for, (i) providing off-chip board-to-board interface, (ii) forming off-chip communication data, (iii) interfacing with multiple connections of the target NoC, (iv) arbitrating between the connections, and (v) controlling the flow of on-chip and off-chip data. What follows describes these units in details.

6.1. Board-to-board interface

The proposed bridge in this paper serves to connect NoC-based systems on the same or separate chips and circuit boards. The chips may be an ASIC or FPGA. In this work we target FPGA technology to prototype the bridge and to prove the concept of bridging.

Typically, FPGA boards support common off-board communication protocols such as PCI [43], Ethernet [44], USB [45], and SPI [46]. Interestingly, all these protocols transfer data serially which corresponds to our design requirements of the bridge. USB and SPI are master–slave protocols which are not applicable to our proposal, since we might have a set of masters and slaves at each side of the bridge. PCI cannot provide wired connections at long distance. Therefore, we build our bridge upon Ethernet protocol as it provides a scalable connection for long distance communication which is suitable for both off-chip and off-board bridging. The bridge architecture utilizes two hardware modules to prototype the Ethernet protocol as shown in Fig. 3(a). The physical layer of OSI protocol stack [15] is implemented by a Xilinx RocketIO transceiver [47]; and the data link layer is realized by Medium Access Control (MAC) module [48,49]. These modules are boards and FPGAs specific. To integrate the bridge in an ASIC, the Ethernet modules should be either redesigned and customized for the target manufacturing technology, or the available individual ICs [50] should be used for this purpose.

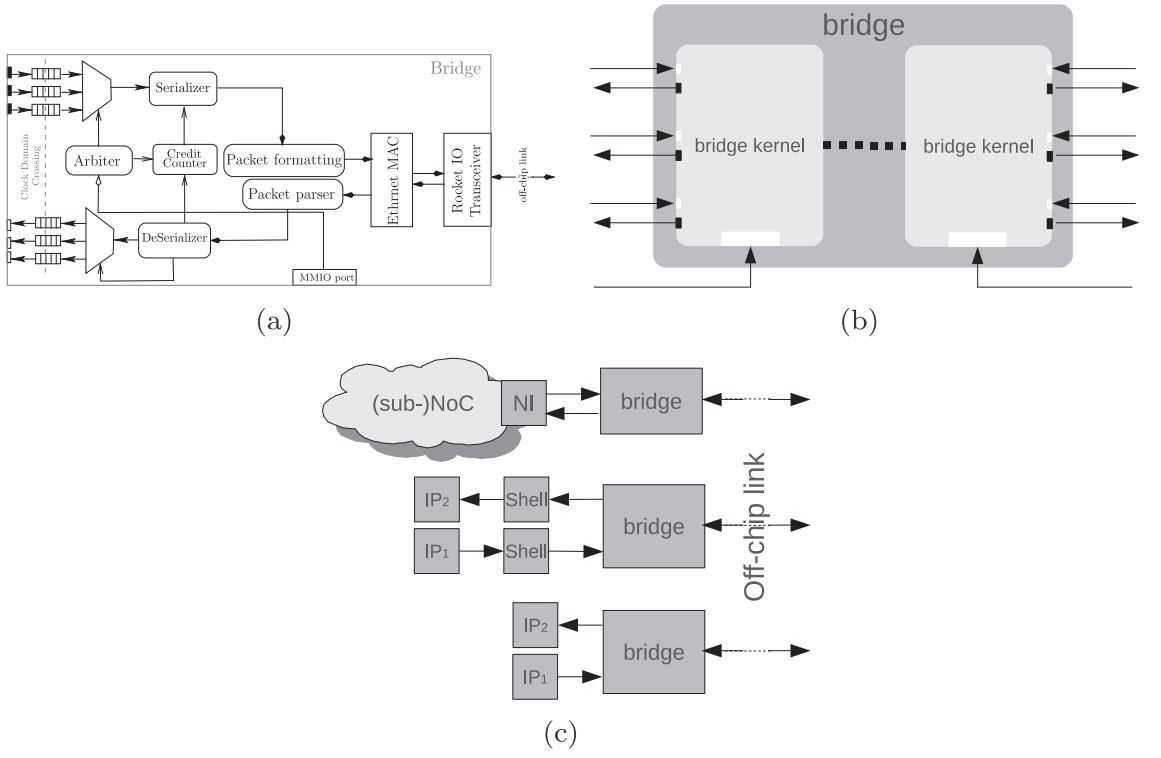

Fig. 3. (a) Bridge kernel architecture, (b) the bridge module, (c) bridge interfacing use-cases.

6.2. Data format

Ethernet transfers data in a specific packet frame format which is illustrated in Fig. 4. There are 38 fixed reserved bytes in a frame for *preamble*, *source* and *destination* MAC address, *type* of packet, *CRC*, and finally an *inter-frame gap*. We refer to them as frame overhead, since only the payload contains useful data.

A payload is made by multiplexing and serializing the streaming data of multiple connections. The bridge forms an Ethernet frame packet by concatenating the payload with the frame overhead. It is done by the *frame former* block in Fig. 3(a).

Moreover, to implement the link-level flow control, the bridge adds the available credits of each connection to the payload. To distinguish between the credits and the actual data, a byte identifier tag is encoded in two least significant bits of a byte as shown in Fig. 4. The bridge starts sending one connection's data by first inserting the connection's number when the byte is tagged as "01". It is then followed by a credit-byte and a 5-byte phit, respectively tagged as "10" and "11". In our target NoC, every phit is formed by 37 bits, which together with byte identifier tag occupies 5 bytes of the payload.

Length of a payload is variable. The longer the length of the payload is, the more efficient is the link utilization. The bridge starts sending a frame whenever data is available in the connection buffers. However, it might occur that after starting a long payload, there would be no data to be sent. Since an Ethernet packet cannot be cut in the middle of transmission, the bridge inserts NULL bytes in such cases. The disadvantage of such a long payload is that CRC can be only done very late after the whole frame received, and therefore, if the CRC is not correct, the bridge should have a buffer with the size of the payload at the transmitter side to re-transmit the last frame. In the current version of the bridge, CRC is checked at the receiver side but the re-transmission feature is left as an option for the future kernel versions.

6.3. Multiple connections

In order to implement QoS, the bridge distinguishes among the NoC connections. The bridge is implemented with dedicated input ports for each NoC connection, which are PPSD ports in Fig. 3(a). The generic architecture of the bridge enables a port to connect to a streaming port of an NI, a shell, or an IP as depicted in Fig. 3(c).

NIs have a dedicated buffer per connection to implement the end-to-end flow control for GT traffic in the NoC. If the bridge is connected to an NI directly, the NI buffers can be also used for the off-chip link-level flow control by the bridge. However, if a shell or an IP is connected directly to the bridge, they might be buffer-less, and therefore virtual channel buffering is required to preserve QoS. The bridge therefore buffers data per connection at the first (last) stage of the transmitter (receiver) side.

Furthermore, the frequency at which the bridge operates on is fixed to 125 MHz and is imposed by the Ethernet MAC and physical modules of the FPGA boards that we used (i.e., Xilinx ML510 and ML605). This may be different from the operation clock frequency of the connected NoC, shell, and IP. Therefore, dual-clock First-in-First-out (FIFO) buffers are required to carry out the clock domain crossing at the NI/shell/IP-bridge connection border.

Per connection buffering in the bridge has also some disadvantages as: (i) it adds extra delays in the data path, and (ii) it is costly both in FPGA and silicon. Therefore, a smart buffer sizing by analyzing the applications traffic is supported, in the same way that it is done by NoCs' design flow for the buffers in NIs and routers of the NoC technology [51].

6.4. Connections arbitration

The bridge arbitrates between the connections at the transmitter side, and serializes the data in the payload of an Ethernet frame. The *arbiter* and *serializer* blocks are respectively responsible for these tasks, as shown in Fig. 3(a). The bridge divides a payload into number of data slots in which it places the arbitrated connection flits. A slot starts with a connection identifier byte, followed by a credit-byte and the flits of the connection. At the receiver side, the bridge parses the payload byte identifier tags (done by *frame parser* block) and directs the flits to a dedicated connection buffer immediately (done by *de-serializer block*).

Any arbitration policy, e.g., Round-Robin (RR) [52], Time-Division-Multiplexing Access (TDMA) [53], may be used by the bridge to select between the connections at the transmitter side. However, to fulfill the QoS design requirement, which implies that no GT traffic should wait for any BE one, an arbitration scheme that ensures independent guarantees per connections, e.g., TDMA, is necessary. Therefore, the default implementation of the bridge uses a TDMA arbiter.

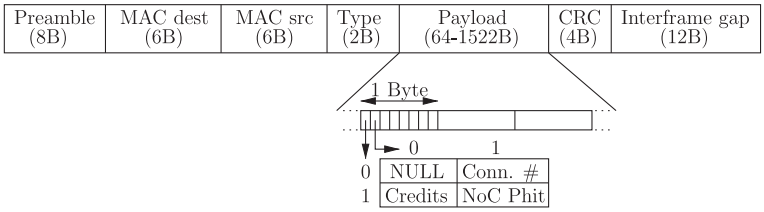

Fig. 4. Ethernet packet frame format with the bridged connections data encoded in the payload.

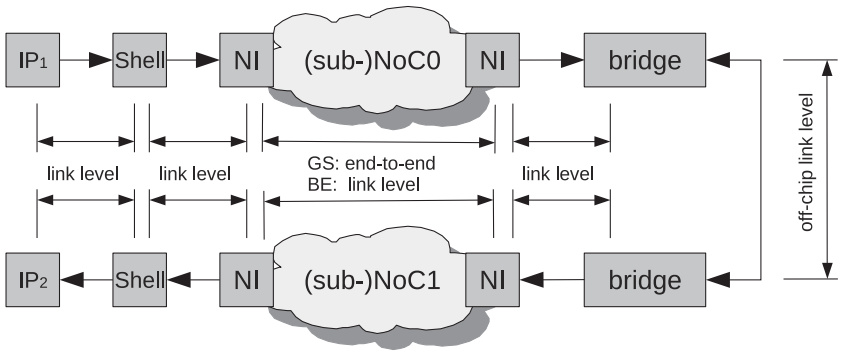

Fig. 5. Flow control mechanisms at different levels.

Since a NoC may be reconfigured at run-time, a connection could be used by different applications that have different QoS requirements. In our case, the bridge arbiter needs to be reprogrammed to alter the connections table at run-time. For this purpose, the bridge is equipped with a memory-mapped input/output (MMIO) port, as shown in Fig. 3(a).

6.5. On/Off-chip flow control

Fig. 5 shows the required flow control at different levels of a bridged system. The NIs that are part of the proposed bridging scheme take care of the global end-to-end flow control in the sub-NoCs for the sake of synchronicity, as explained in Section 4.

A flow control mechanism for the off-chip link of the bridge is also needed. We use a credit-based technique [54] with one credit counter only at the transmitter side of the bridge, for each connection buffer. The counter is initialized with the size of corresponding buffer at the receiver side, and it is decremented every time that a phit is transmitted. In the receiver side, another counter keeps the number of received phits that have been out of buffers. This value is then sent back to the transmitter side as a credit-byte. As soon as the credit is received, the credit counter is updated. The credit transmission from receiver side is guaranteed by first sending the credit of corresponding connection in every slot so that no connection starves due to the lack of credits. Hence, the bridge is deadlock free.

7. Software architecture

Typically, NoC-based SoCs require a run-time configuration scheme [17]. A processor core which is usually on the same chip as the NoC is locally responsible to perform the configuration. This processor is called *host*. In the presence of an off-chip connection, the host may be an external IP such as a Personal Computer (PC) or a local host of another SoC on a separate chip.

In this section we first briefly present a basic interconnect configuration scheme, which is proposed in [17], and we show how an on-chip host configures the NoC. Afterward, we present how the bridge extends this scheme to configure multi-chip NoC-based systems while the host may be either on-chip or off-chip. The novelty of this technique is in that it uses the interconnect of a local (sub-) system, which is directly connected to a host, to configure remote interconnected (sub-) systems via the bridge.

7.1. Basic software scheme

The NoC configuration is the process of setting up, modifying, or tearing down logical connections between master and slave IPs. This process is performed by programing routing scheduling tables in routers and network interfaces, and address layout tables in buses. Each of these modules should have a memory-mapped input/output port (MMIO) for this purpose.

In this work, we consider an interconnect that does not require the configuration of routers, hence, only NIs and buses are programmable. Fig. 6(a) illustrates such a system interconnecting an internal host, a master and a slave IP. The config-bus enables the internal host to access distributed memory-mapped locations. We group the config-bus and the internal host in a logical component, namely Local Host (LH). This component is always located on the same chip as the target NoC.

The LH can access MMIO ports of NIs and buses from a dedicated port. This port is called *rmt* in Fig. 6(a). To enable this access, a path from NI_c to a config port of target NI is required. The configuration software executing on the host sets up this path by sending a write command to *lcl* port of LH to program NI_c table. We define this command as `write(`LH_{lcl}, `path(`LH_{rmt} `→target))`, where the `target` is the config port of NI_c.

LH performs NoC configuration in two steps, (i) initialization step, in which it opens a response channel from all NIs (i.e., NI_m and NI_s in Fig. 6(a)) to NI_c to enable sending back configuration end-to-end flow control, and (ii) connection set-up step, in which the host sets up a functional connection between master and slave.

A connection is established between a *master* and a *slave* IP, i.e., a connection is opened, by setting up a request and a response channel. To set up a channel between IP *A* and *B* in the system, LH issues a sequence of write commands to program

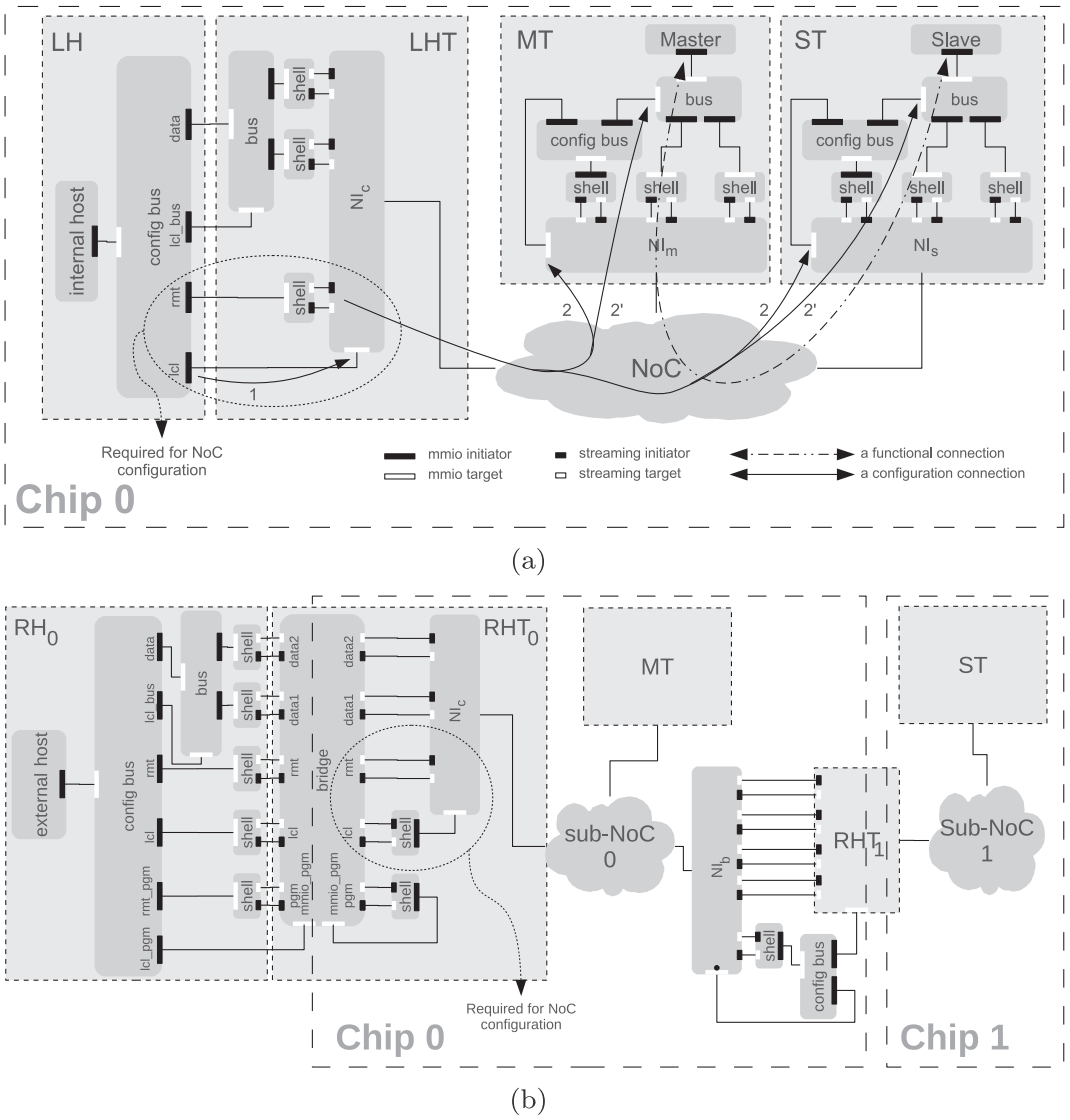

Fig. 6. Configuration architecture of (a) an on-chip interconnect using a local host, (b) an extended-on-chip interconnect using the bridge inside Remote-Hosting Tile (RHT).

NIs and buses. The software API executing on the host implements it as `open_channel (A,B)` and it is described in Algorithm 1. Every write in the algorithm corresponds to a numbered arrowed operation in Fig. 6(a), and it is illustrated by the numbers before the commands. Operation 1 is to program NI_c and open/close a request channel from *rmt* port to a target. Operation 2 and 2′ are to configure the target NI_m, NI_s, and buses.

Algorithm 1. Open a channel between A and B

```
open_channel (A,B) {
  if A = LH then
    (1) write(LH_lcl, path(LH_data →A));
  else
    (1) write(LH_lcl, path(LH_rmt →A));
    (2) write(LH_rmt, path(A→B));
    (2′) write(LH_rmt, bus address layout);
    (1) write(LH_lcl, ∅);
  end
}
```

Algorithm 2 uses `open_channel` to implement an API (`open_connection (A,B)`) that establishes a connection between two IPs. Applying this command to every master and slave pair in the design, a use-case application can be (re-) configured.

Algotithm 2. Open a connection between A and B

```
P = {A,B} where p_i ∈ P and 1 < i ⩽ |P|;
open_connection (P) {
  open_channel (p_1, p_2);
  open_channel (p_2, p_1);
}
```

7.2. Extended-interconnect software scheme

Consider that the system in Fig. 6(a) is partitioned into two sub-systems, as illustrated in Fig. 6(b). The master tile (MT) and the slave tile (ST) are connected to different NoCs implemented on separate chips. Although in this example the host is off-chip with respect to the two sub-NoCs in Fig. 6(b), it can optionally be located locally on one of the chips. Here, a new configuration scheme is essential to enable the external host (i.e., off-chip host) to configure connections over both sub-NoC_0 and sub-NoC_1, as local and remote sub-systems, respectively.

The core module of the new configuration scheme is the Remote-Hosting Tile (RHT). As illustrated in Fig. 6(b), the RHT consists of a bridge module, a NI_c, and two shells. The RHT serves two purposes. First, it implements an off-chip connection to the SoC by the bridge, and second, it enables remote configuration of the NoCs by providing the required links (via *lcl* and *rmt* ports) to the local NI_cs of remote sub-NoCs.

In Fig. 6(b), there are two RHTs. One, RHT_0, connects the remote host (i.e., RH_0) to the sub-NoC_0. RH_0 consists of an external host, buses and shells, and plays the same role of LH in Fig. 6(a), namely it sends proper write commands to its *lcl* and *rmt* ports. The other remote-hosting tile (i.e., RHT_1) interconnects two sub-NoCs to provide functional data connections between master and slave tiles. RHT_1 enables configuration of sub-NoC_1, which is a remote sub-NoC, from RH_0's point of view.

Algorithm 3. Open a channel between A and B

```
open_channel (A,B,C) {
  if C ≠ ∅ then
    write(C_lcl, path(C_rmt → A));
  end
  if A ≠ LH/RH and A ≠ C then
    write(C_rmt, path(A→B));
    write(C_rmt, bus address or bridge arbiter layout);
    write(C_lcl, ∅);
  end}
```

Algorithm 4. Open a connection between A and B

```
open_connection (P,C_0){
  for i ← 1 to |P| − 1 do
    open_channel (p_{i−1}, p_i, C_{i−1});
    open_channel (p_i, p_{i−1}, C_{i−1});
    C_i = p_i;
  end
}
```

The RHTs provide the underlying infrastructure for the multi-NoC communication and configuration, and the bridge inside them provides transparent connections to the configuration application executing on the external host. The basic software API defined earlier in this section is not directly applicable to the multi-chip bridged system. The reason is that a functional connection between a master and slave that are located on different chips, goes across one or more RHTs. Therefore the configuration software on one hand should deal with configuration of two (or more) sub-NoCs instead of one, and on

the other hand it should use the proper RHT for a target sub-NoC. For this purpose, we extend `open_channel` and `open_connection` commands with a new argument. The argument is the hosting tile, i.e., RHT or LHT, that the host uses to perform configuration. The new commands are presented in Algorithms 3 and 4, respectively.

Furthermore, a design is not always as simple as the systems illustrated in Fig. 6. The bridge enables us to build complex systems by interconnecting number of sub-NoCs and having master and slave IPs interfacing to any of them. Therefore, a set of generic configuration software API is needed to support all multi-chip designs with any level of complexity. In what follows we first show various possible master–slave pair connections in a multi-NoC designs, and then we propose a new configuration API.

Fig. 7(a) and (b) illustrate the abstract connection models of the simple systems presented in Fig. 6. The host may be implemented as either local host (LH) or remote host (RH). In the design of Fig. 6(b), master and slave tiles can also interface both to one remote sub-NoC, as it is modeled in Fig. 7(c). A more general complex design, assuming more than one remote sub-NoCs, and, master and slave tiles interface with separate sub-NoCs, is illustrated in Fig. 7(d).

We propose a generic connections graph in Fig. 8(a). We define the connection graph as $G = (V, E)$, where V={LH, RH, LHT, RHT, MT, ST} is a set of vertices, and E is a set of communication edges. A host that may be either RH or LH is the root-vertex of G, and, master tile (MT) and slave tile (ST) are two end-vertices. The edges are the applications' functional connections between the tiles, where dashed edges represent an arbitrary sequence of edge-RHT. As shown in Fig. 8(a), we define three paths of P_{HM}, P_{HS}, P_{MS} that represent the paths from RH/LH to MT, from RH/LH to ST, and from MT to ST, respectively.

Algorithm 5. Generic configuration procedure to set up a connection between a master and slave

assuming:
P_{MS} = {path starting from master to slave}
P_{HM} = {path starting from host to master}
P_{HS} = {path starting from host to slave}
P_0 = first member of P, and P_L = last member of P.
begin
1. open_connection $((P_{HM} \cap P_{HS}), \emptyset)$;
2. open_connection $(\{(P_{HM} \cap P_{HS})_L, (P_{MS} \cap P_{HM})_L\}, (P_{HM} \cap P_{HS})_L)$;
3. open_connection $((P_{MS} \cap P_{HM}), (P_{MS} \cap P_{HM})_L)$;
4. open_connection $(\{(P_{HM} \cap P_{HS})_L, (P_{MS} \cap P_{HS})_0\}, (P_{HM} \cap P_{HS})_L)$;
5. open_connection $((P_{MS} \cap P_{HS}), (P_{MS} \cap P_{HS})_0)$;
6. open_connection $(\{(P_{MS} \cap P_{HM})_L, (P_{MS} \cap P_{HS})_0\}, (P_{HM} \cap P_{HS})_L)$;
end

P_{MS} denotes the functional connection between MT and ST that should be established by the host. For this purpose, we propose to create six intermediate logical connections. The connections are illustrated in Fig. 8(b) that corresponds to the graph of Fig. 8(a). Connection 1 enables the host to access remotely to the common sub-NoC that can access to both MT and ST. To access to MT, it needs connection 2 and 3, and similarly to ST connection 4 and 5 are needed. Finally, the host completes the connection between MT and ST by establishing connection 6. Algorithm 5 uses the extended `open_connection` command to make the connections of Fig. 8(b) in the six steps.

To summarize, we proposed a new software API for configuration of multi-chip NoC-based systems in this section. The proposal used the hardware tiles which includes the bridge for remote (sub-) NoCs' accesses, and the new generic software API enables configuration of complex NoC-based designs.

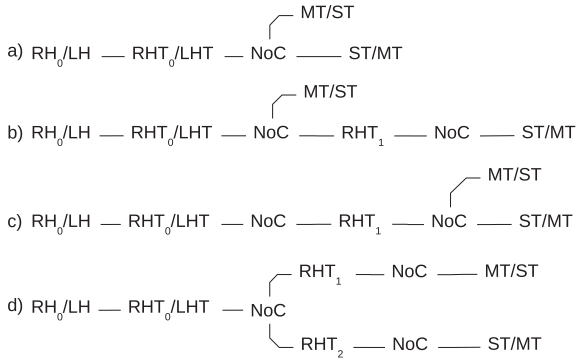

Fig. 7. connection models of single- and multi-chip NoC-based systems with different combinations of master and slave tile interfaces.

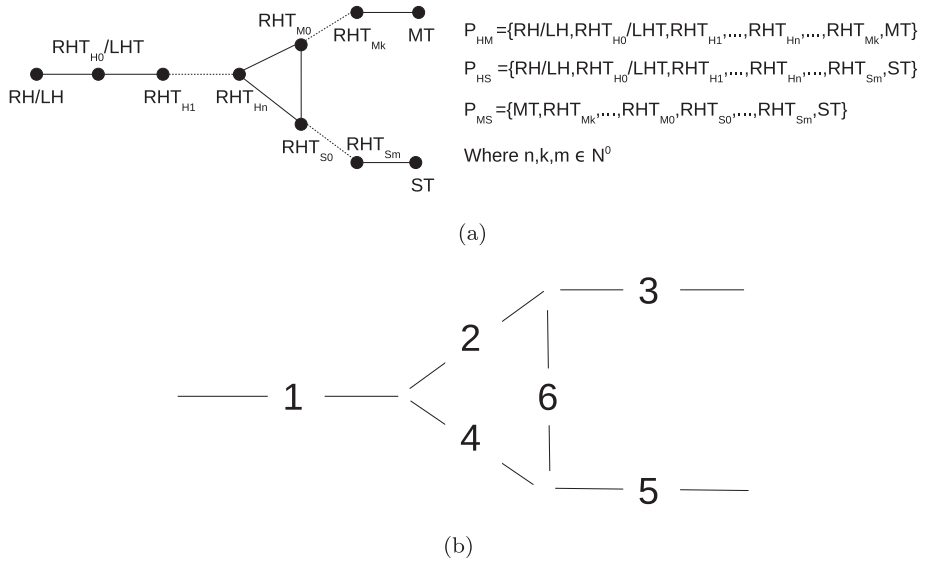

Fig. 8. (a) Generic configuration connections graph, and (b), logical intermediate connections graph for a multi-chip NoC-based design.

8. Discussion and experimental results

In this section we first exercise the standalone bridge to evaluate its area cost when implemented on an FPGA, and to assess its performance under variable traffic loads. Second, we use the bridge in a NoC-based system to connect two sub-systems implemented on two separate FPGA chips. Using this setup, we show system level performance results of the bridge. The results is obtained from experimenting the system for interconnect configuration with various applications' data transfer. The results prove that the bridge preserves traffic QoS over the NoC-based systems.

8.1. Standalone bridge experiments

The bridge is synthesized for Virtex 5 and Virtex 6 FPGAs with Xilinx tools, and it is implemented on the ML510 and ML605 emulation boards, respectively. The embedded tri-mode Ethernet MAC Wrapper, specifically designed for those Xilinx FPGAs, is used as the MAC layer module of the bridge [48,49]. 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 synthesis area results are given in Table 2. As seen, the area cost is different when targeting different FPGA technologies, however percentage-wise the low cost of our implementation is comparable with the work presented in [28]. In our case, the bridge is synthesized to handle four connections with the buffer sizes of 64 phits (64×37 bits) per connection.

To assess data transfer latency between the first stage connection buffers at the transmitter side and the last stage buffers at the receiver side of the bridge, we set up an experimental platform illustrated in Fig. 9. The bridge arbiter uses TDMA policy with the table size of 16 slots. The bridge is connected to a Microblaze processor, μB, that generates data traffic based on the given rates. The processor accesses an external RAM in two different ways, through a bridged connection or directly through a PLB bus, which are illustrated as Path 1 & 2 in Fig. 9, respectively. Knowing the latency of all the components, we obtain the data transfer latency of the bridge with this set-up.

Fig. 10(a) presents the latency versus the number of TDMA slots allocated to one connection. The trend in the figure indicates that the latency decreases when the number of assigned slots to a connection increases. The reason is that in each cycle of TDMA table more data is transferred by the bridge and the waiting time of transactions in the transmitter buffer becomes shorter. However, the latency cannot be lower than 230 cycles (in average) which is much higher than the lowest average 40-cycle latency of the work presented in [28,4]. This is due to the reserved slots in the TDMA table plus the latency of the off-

Table 2
The Bridge synthesis results.

Cell	Virtex5 usage	Virtex6 usage
Slices as LUTs	722	2142
Slices as flip flops	389	559
Slices as memories	204	404

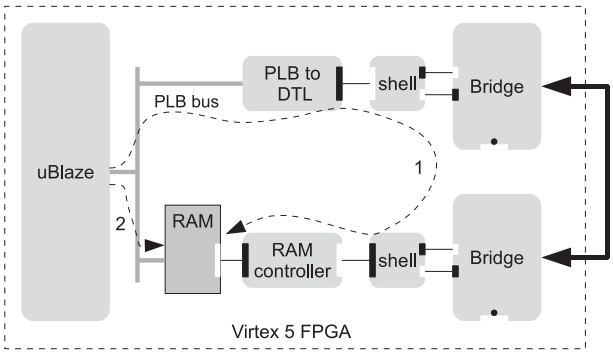

Fig. 9. The experimental platform for assessing performance of the standalone bridge.

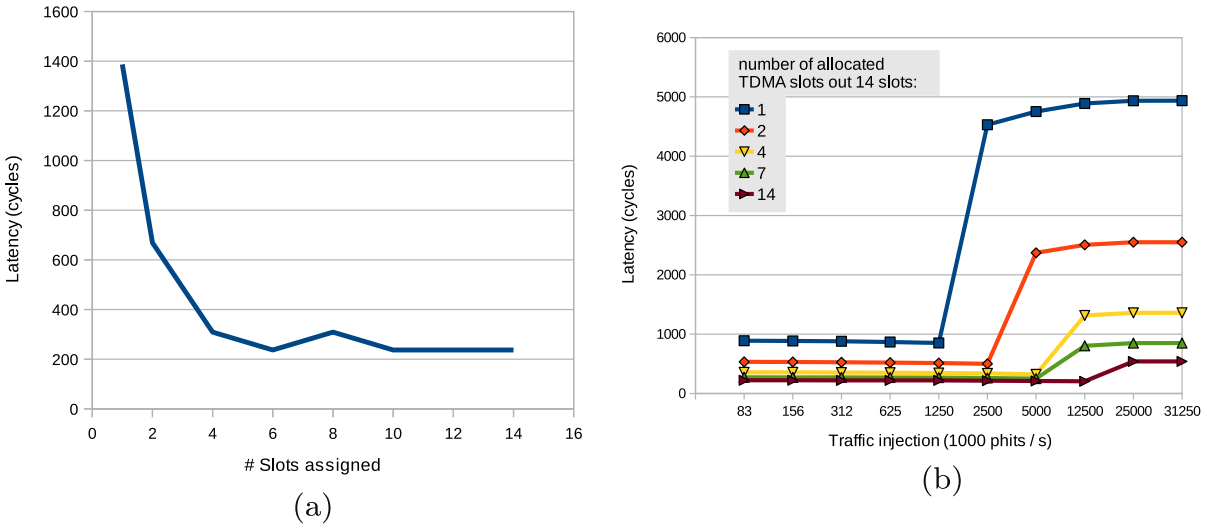

Fig. 10. (a) Latency vs. number of allocated slots to a connection, (b) Latency vs. throughput for one connection with variable slots assigned.

chip link which is approximately 220 cycles. The reserved slots are to send the frame overhead of the Ethernet packet, and in our case they are the last slots of the table. The reason that the latency of 8-slot allocation does not follow the general trend of Fig. 10(a) is due to the fact that for this value, the payload of the Ethernet frame was entirely transmitted in the middle of TDMA wheel, but the frame transfer finishes once the frame overhead is sent in the reserved slots at the end of the table. Hence, the ready data in the buffer should wait therefore till a new cycle of TDMA table.

Fig. 10(b) shows the data transfer latency of a connection versus the throughput of the bridge for different number of assigned TDMA slots. The initial flat region in the graphs shows that the connection does not utilize its entire allocated guaranteed bandwidth. Since the waiting time outside the buffers is not included here, increasing the offered load beyond the guaranteed bandwidth budget increases the latency to a finite maximum. To summarize, allocating more slots to a connection increases the bandwidth and lowers the latency.

Here, we calculate the ideal throughput of the bridge in order to compare with the empirical results in Fig. 10(b). The bridge uses the maximum frame length of $P_{eth} = 1500$ bytes and $N_s = 10$ slots per frame. Therefore each slot has $s_{slot} = 150$ bytes. Thus, the number of phits per slot is:

$$phits_{slot} = \frac{150 - 2}{5} = 29\ phits$$

The minus two is due to the connection byte and the credit byte. Finally the link utilization is:

$$\eta = \frac{N_s \times phits_{slot} \times s_{phit}\ (bits)}{L_{eth\_max} \times 8\ (bits)} = \frac{10 \times 29 \times 37}{1542 \times 8} = 86.98\%$$

Thus, the ideal throughput would be 870 Mbps if the bridge is using the full bandwidth of 1Gbit Ethernet. As in above calculations we assumed that the whole bandwidth (i.e., the whole payload) is dedicated to one connection, we compare the ideal throughput with the 14-slot graph. The ideal throughput approximately equals to 23500 cycles (1000 phit/S) that corresponds to a point slightly before saturation at 14-slot graph in Fig. 10(b) and it is in line with the experimental results.

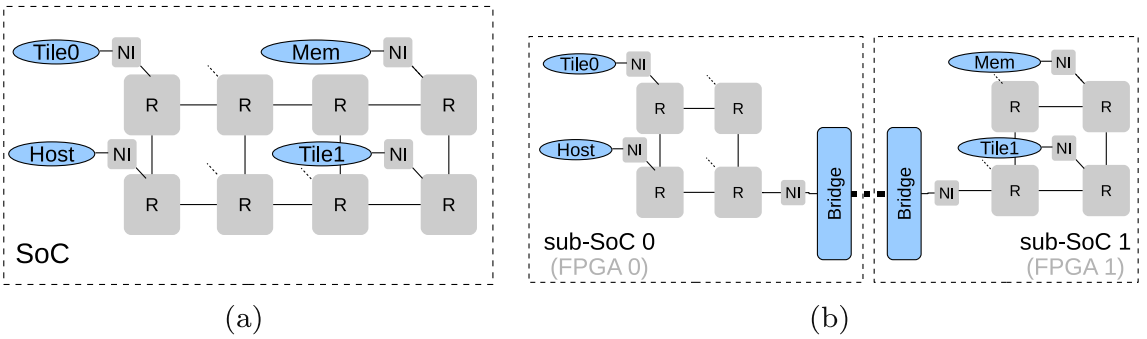

Fig. 11. (a) A NoC-based system-on-chip, (b) a multi-FPGA prototype of bridged partitioned NoC-based sub-SoCs.

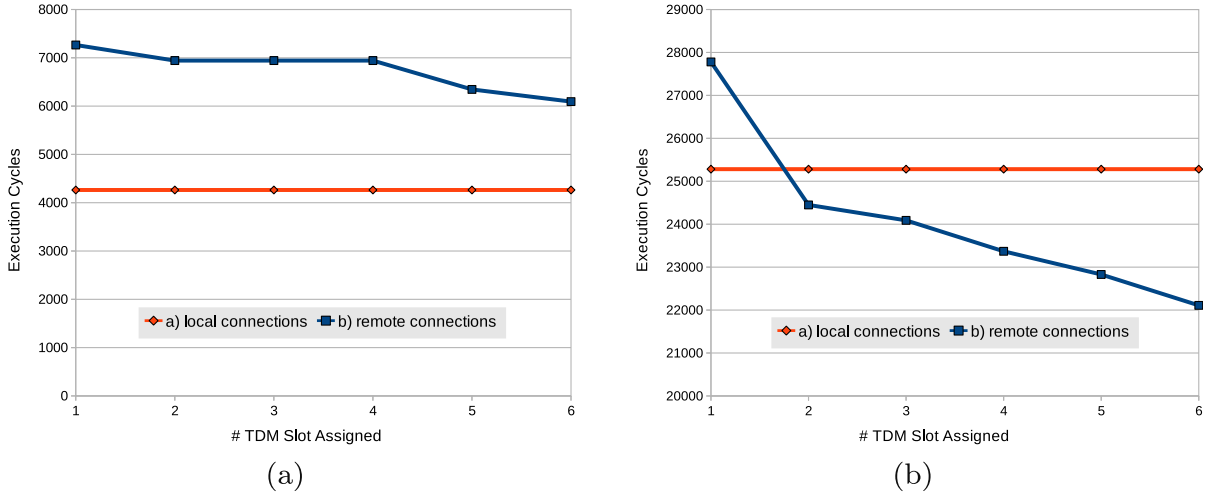

Fig. 12. The Experimental results of, extended-interconnect configuration (a) initialization step, (b) connection set-up step.

8.2. System level experiments

After presenting the cost and performance of the individual bridge, in the rest of this section we demonstrate a real multi-FPGA system, which is built by using the bridge. The original SoC is illustrated in Fig. 11(a). There are two processor tiles, a memory instance, and a host. This system is partitioned into two sub-systems, each of which prototyped on a separate Xilinx ML605 board, as shown in Fig. 11(b). This simple set-up helps us to show more clearly how the bridge preserves traffic QoS.

8.2.1. Interconnect configuration

The time that it takes to configure the interconnect is presented in Fig. 12, where 12(a) shows the results for initialization step and 12(b) for connection set-up step. In this example, we set up 7 connections that cross the bridge. We have varied the number of TDMA slots allocated to *lcl* and *rmt* ports of the bridge, and we measured the configuration execution time in number of cycles.

In this set-up, we have 4 NIs available at each sub-SoC to be initialized. The remote initialization step takes longer than the local one, as shown in Fig. 12(a). However, the remote connection set-up step takes less time than local step when enough bandwidth is available. The reason is that the host should configure both master and slave NIs in local configuration step, whereas in the remote one the bridge itself is the master and it is not needed to configure itself, and therefore it performs less configuration operations.

8.2.2. Applications traffic QoS

To mimic the behavior of a communication intensive application with both GT and BE traffic QoS, we use software implemented traffic generators on Microblaze processor tiles following the topology presented in Fig. 11(b). Five chip-crossing connections are set-up between the cores. The QoS requirements and number of allocated bridge TDMA slots per connection are detailed in Table 3.

Table 3
Connections.

Conn. #	Master → slave	# of allocated TDMA slots	QoS
0	Tile0 → Tile1	4	GT
1	Tile0 → Tile1	1	BE
2	Tile0 → Mem	4	GT
3	Tile1 → Tile0	1	BE
4	Tile1 → Tile0	4	GT

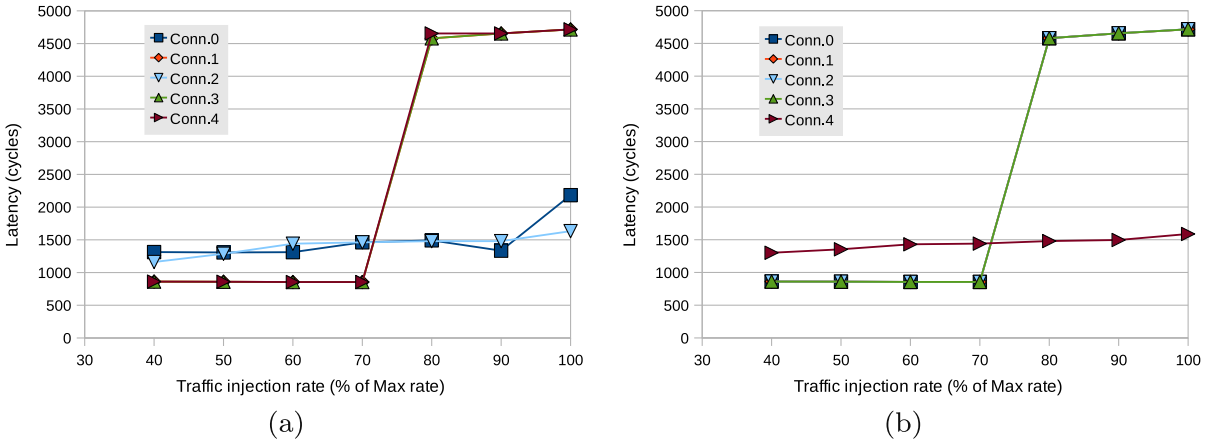

Fig. 13. Latency vs. throughput for the connections initiated in the example platform from the left sub-system (a) and from the right sub-system (b).

Fig. 13(a) shows the latency versus throughput for the traffic of channels initiated from the sub-SoC_0. These are request channels of Connection 0, 1 and 2, and response channels of Connection 3 and 4. We vary the throughput with percentaged steps of the maximum bandwidth offered by the bridge to a connection. The maximum bandwidth of a connection is derived from the throughput results presented in Fig. 10(b) for corresponding number of allocated slots per connection.

The request channels of Connection 0 and 2 have the lowest latency for the higher bandwidth, since they are allocated 4 TDMA slots and for the other connections, each is assigned one slot. As it is illustrated in Fig. 13(a), the average latency of traffic for these two connections are guaranteed to less than 1500 cycles for up to 90% of the maximum traffic injection rate which is 5 M phits/s. However, since the response channels of Connection 3 and 4 require BE QoS, their latency and throughput is not guaranteed for high traffic injection rates, and after 70% of the maximum injection rate, their average latency jumps from less than 1000 cycles to around 4500 cycles.

Similarly, Fig. 13(b) shows the results for the traffic of channels initiated from the sub-SoC_1. Here, the lowest latency and higher bandwidth also belongs to the GT traffic of Connection 4, while the traffic 0, 1 and 2 are the response channels of their corresponding connections.

To summarize, we have shown that the bridge preserves QoS, namely GT or BE, requested by the connections while transferring the traffic across a multi-chip/FPGA system. Note however that, this does not imply that the connection's maximum bandwidth and latency are unchanged. Only the type of traffic QoS (GT/BE) is guaranteed to be unchanged. Depending on the number of bridged connections, the maximum bandwidth of each connection may be decreased and the latency may be increased. In any case, the global memory space is transparent to all applications across all the bridged (sub-) systems.

9. Conclusions

In this paper we proposed a generic, efficient off-chip bridging scheme for SoCs that are implemented on separate silicon or FPGA chips. The scheme is compatible with NoCs technology, which is commonly utilized in recent embedded systems. We have investigated the protocol stack of an on-chip interconnect to determine the best layer of implementing the bridge on possible links of the interconnect. The proposal is a scheme at the transport layer of the stack. At this layer, the bridge fulfills the requirements which are identified as: (i) seamlessly extending memory space over the sub-NoCs, (ii) decoupling of bridged systems, (iii) preserving applications quality of service, (iv) being cost efficient and having high performance.

We presented a hardware architecture for the bridge and a software infrastructure needed to configure the interconnects of the bridged systems. For this purpose, we extended a basic configuration scheme by using the bridge, and we proposed a new architecture for software configuration API. This is demonstrated practically by the experiments that have evaluated the bridge implementation under variable traffic loads. The results showed that the bridge preserves the traffic quality of service

as it can provide throughput and latency guarantees to traffic. Moreover, the bridge is synthesized for two Xilinx FPGA chips and the area cost is less than 1% of the FPGA resources.

Future research work includes a SoC partitioning technique to automatically generate the bridged sub-systems. This technique is required to be integrated in automated NoCs design flow.

References

[1] C. Liu, I. Ganusov, M. Burtscher, S. Tiwari, Bridging the processor-memory performance gap with 3D IC technology, IEEE Design and Test of Computers 22 (2005) 556–564.
[2] F. Steenhof, H. Duque, B. Nilsson, K. Goossens, R.P. Llopis, Networks on chips for high-end consumer-electronics TV system architectures, in: Proc. DATE, 2006, pp. 148–153.
[3] A. Kulmala, E. Salminen, T. Hamalainen, Evaluating large system-on-chip on multi-FPGA platform, Embedded Computer Systems: Architectures, Modeling, and Simulation (2007) 179–189.
[4] A.-M. Kouadri-Mostefaoui, B. Senouci, F. Petrot, Scalable multi-FPGA platform for networks-on-chip emulation, in: Proc. ASAP, 2007, pp. 54–60.
[5] S. Hauk, Multi-FPGA Systems, Ph.D. Thesis, University of Washington, 1995.
[6] K. Goossens, B. Vermeulen, A.B. Nejad, A high-level debug environment for communication-centric debug, in: Proc. DATE, 2009, pp. 202–207.
[7] A. Shacham, K. Bergman, L. Carloni, On the design of a photonic network-on-chip, in: Proc. NOCS, 2007, pp. 53–64.
[8] M. Stepniewska, A. Luczak, J. Siast, Network-on-Multi-Chip (NoMC) for Multi-FPGA Multimedia Systems, in: Proc. Conf. on Digital System Design (DSD), 2010.
[9] S. Furber, S. Temple, A. Brown, On-chip and inter-chip networks for modeling large-scale neural systems, in: Proc. ISCAS, 2006, p. 4.
[10] W.J. Dally, B. Towles, Route packets, not wires: on-chip interconnection networks, in: Proc. DAC, 2001, pp. 684–689.
[11] K. Goossens, A. Hansson, The Aethereal network on chip after ten years: goals, evolution, lessons, and future, in: Proc. DAC, 2010.
[12] M. Millberg, E. Nilsson, R. Thid, A. Jantsch, Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip, in: Proc. DATE, 2004.
[13] T. Bjerregaard, The MANGO clockless network-on-chip: concepts and implementation, IMM, Danmarks Tekniske Universitet, 2005.
[14] E. Bolotin, I. Cidon, R. Ginosar, A. Kolodny, QNoC: QoS architecture and design process for network on chip, Journal of Systems Architecture 50 (2004) 105–128.
[15] J.D. Day, H. Zimmerman, The OSI reference model, in: Proceedings of the IEEE, vol. 71, 1983, pp. 1334–1340.
[16] A. Hansson, K. Goossens, An on-chip interconnect and protocol stack for multiple communication paradigms and programming models, in: Proc. CODES+ISSS, 2009.
[17] A. Hansson, K. Goossens, Trade-offs in the configuration of a network on chip for multiple use-cases, in: Proc. NOCS, 2007, pp. 233–242.
[18] M. Lee, C. Chen, Multi-chip module, 2002. US Patent 6,388,313.
[19] J. Darnauer, P. Garay, T. Isshiki, J. Ramirez, W. Wei-Ming Dai, A field programmable multi-chip module (FPMCM), in: Proc. FCCM, 1994, pp. 1–10.
[20] J. Darnauer, T. Isshiki, P. Garay, J. Ramirez, V. Maheshwari, W. Dai, A silicon-on-silicon field programmable multichip module (FPMCM) integrating FPGA and MCM technologies, IEEE Transactions on CPMT 18 (1995) 601–608.
[21] K. Takahashi, M. Sekiguchi, Through silicon via and 3-D wafer/chip stacking technology, in: Proc. Symp. on VLSI Circuits, 2006, pp. 89–92.
[22] E. Beigne, P. Vivet, Design of on-chip and off-chip interfaces for a GALS NoC architecture, in: Proc. ASYNC, 2006, pp. 10–183.
[23] S. Evain, J.-P. Diguet, D. Houzet, NoC design flow for TDMA and QoS management in a GALS context, EURASIP Journal on Embedded Systems (2006) 4-4.
[24] P. Del Valle, D. Atienza, I. Magan, J. Flores, E. Perez, J. Mendias, L. Benini, G. De Micheli, A complete multi-processor system-on-chip FPGA-based emulation framework, in: Proc. IFIP VLSI-SoC, 2006, pp. 140–145.
[25] C. Chang, J. Wawrzynek, R.W. Brodersen, BEE2: a high-end reconfigurable computing system, IEEE Design and Test of Computers 22 (2005) 114–125.
[26] P. Liu, C. Xiang, X. Wang, B. Xia, Y. Liu, W. Wang, Q. Yao, A NoC emulation/verification framework, in: Proc. ITNG, 2009, pp. 859–864.
[27] X. Li, O. Hammami, Multi-FPGA emulation of a 48-cores multiprocessor with NOC, in: Proc. IDT, 2008, pp. 205–208.
[28] Kouadri-Mostefaoui, Abdellah-Medjadji, B. Senouci, F. Petrot, Large scale on-chip networks: An accurate multi-FPGA emulation platform, in: Proc. DSD, 2008, pp. 3–9.
[29] B.P. Kommineni, R. Srinivasan, R. Holsmark, A. Johansson, S. Kumar, Modeling and evaluation of a NoC-internet interface, in: Swedish System on Chip Conference, Bstad, April 13–14, 2004.
[30] Y. Yin, S. Chen, Design and implementation of a inter-chip bridge in a multi-core SoC, in: Proc. DTIS, 2009, pp. 102–106.
[31] A. Patel, C.A. Madill, M. Saldana, C. Comis, R. Pomes, P. Chow, A scalable FPGA-based multiprocessor, in: Proc. FCCM, 2006, pp. 111–120.
[32] Arteris, A comparison of network-on-chip and busses, White paper, 2005.
[33] L. Benini, G. de Micheli, Powering Networks on Chips, in: Int'l Symposium on System Synthesis (ISSS), 2001.
[34] T. Bjerregaard, S. Mahadevan, A survey of research and practices of network-on-chip, ACM Computer Survey 38 (2006).
[35] P. Martin, Design of a virtual component neutral network-on-chip transaction layer, in: Proc. DATE, 2005.
[36] M. Millberg et al., The Nostrum backbone – a communication protocol stack for networks on chip, in: Proc. International Conference on VLSI Design, 2004.
[37] S. Murali, M. Coenen, A. Radulescu, K. Goossens, G. De Micheli, Mapping and configuration methods for multi-use-case networks on chips, in: Proc. ASP-DAC, 2006, pp. 146–151.
[38] Open Core Protocol (OCP) Specification, 2003. <www.ocpip.org>.
[39] DTL, Device Transaction Level (DTL) Protocol Specification. Version 2.2, Philips Semiconductors, 2002.
[40] AMBA AXI Protocol Specification, ARM, 2003.
[41] D. Wingard, Socket-based design using decoupled interconnects, in: Interconnect-Centric Design for Advanced SoC and NoC, Kluwer, 2004, pp. 367–396.
[42] W. Dally, B. Towles, Principles and Practices of Interconnection Networks, Morgan Kaufman, 2004.
[43] T. Shanley, D. Anderson, PCI System Architecture, second ed., Addison-Wesley Longman Publishing Co., Inc., 1995.
[44] R.M. Metcalfe, D.R. Boggs, Ethernet: distributed packet switching for local computer networks, Communications of the ACM 19 (1976) 395–404.
[45] D. Anderson, D. Dzatko, Universal Serial Bus System Architecture, second ed., Addison-Wesley Longman Publishing Co., Inc., 2001.
[46] X. Tian, J. Li, Y. Fan, X. Yu, J. Liu, Design and implementation of SPI communication based-on FPGA, Advanced Materials Research 291 (2011) 2658–2661.
[47] RocketIO Transceiver User Guide, Xilinx, 2007.
[48] Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC, Xilinx, 2011.
[49] Virtex-6 FPGA Embedded Tri-Mode Ethernet MAC, Xilinx, 2011.
[50] Precision PHYTER - IEEE 1588 Precision Time Protocol Transceiver, National Semiconductor, 2010.
[51] M. Coenen, S. Murali, A. Ruadulescu, K. Goossens, G. De Micheli, A buffer-sizing algorithm for networks on chip using TDMA and credit-based end-to-end flow control, in: Proc. of CODES+ISSS, 2006, pp. 130–135.
[52] E. Shin, I. Mooney, V.J., G. Riley, Round-robin arbiter design and generation, in: Proc. Symp. on System Synthesis, 2002, pp. 243–248.
[53] A. Schranzhofer, J.-J. Chen, L. Thiele, Timing analysis for tdma arbitration in resource sharing systems, in: Proc. RTAS, 2010, pp. 215–224.
[54] J. Billington, S. Saboo, An investigation of credit-based flow control protocols, in: Proc. SIMUTools, 2008, pp. 34:1–34:10.