SAMOS, II(), 275–299 (2002)

Communication Services for Networks on Chip

Andrei Rădulescu and Kees Goossens

Philips Research Laboratories, Eindhoven, The Netherlands
Email: andrei.radulescu|kees.goossens@philips.com

Abstract

Networks are emerging as a possible solution for future on-chip interconnects. In this chapter, we show how networks on chip (NoC) are similar to and differ from both off-chip networks (e.g., computer networks) and current on-chip interconnects (e.g., buses). We re-examine their communication services in the context of NoCs. To enable a clean separation between the NoC and IP blocks, we provide services that abstract from network implementations. We define a request-response transaction model similar to bus protocols, to make our approach backward compatible. To exploit the full power of NoCs, we also provide connection-oriented communication with differentiated services. Examples are bandwidth guarantees, transaction orderings, and end-to-end flow control.

Key Words: Networks on chip, on-chip buses, computer networks, communication services, protocol stack, transaction, connection

I. Introduction

Networks on chip (NoC) have received considerable attention recently as a solution to the interconnect problem in highly-complex chips [3–5, 7–9, 15, 20, 23]. The reason is twofold. First, NoCs help resolve the electrical problems in new deep-submicron technologies, as they structure and manage global wires [3–5, 7, 8]. At the same time they share wires, lowering their number and increasing their utilization [7, 8]. NoCs can also be energy efficient and reliable [4], and are scalable compared to buses [9]. Second, NoCs also decouple computation from communication, which is essential in managing the design of billion-transistor chips [14, 23]. NoCs achieve this decoupling because they are traditionally designed using protocol stacks [22], which provide well-defined interfaces separating communication service usage from service implementation [5, 23].

Using networks for on-chip communication when designing systems on chip (SoC), however, raises a number of new issues that must be taken into account. This is because, in contrast to existing on-chip interconnects (e.g., buses, switches, or point-to-point wires), where the communicating modules are directly connected, in a NoC the modules communicate remotely via network nodes. As a result, interconnect arbitration changes from centralized to distributed, and issues like out-of order transactions, higher latencies, and end-to-end flow control must be handled either by the intellectual property block (IP) or by the network.

Most of these topics have been already the subject of research in the field of computer networks [25] and parallel machine interconnect networks [6]. However, on-chip networks have different properties (e.g., tighter link synchronization) and constraints (e.g., higher memory cost) leading to different design choices, which ultimately affect the network services.

In this chapter, we compare NoCs and off-chip networks showing both their similarities and differences. We also explore the differences between NoCs and existing on-chip interconnects. We present an interface which takes these issues into consideration. Our interface is aimed at being similar to a split-transaction bus interface, such as VCI [26], OCP [17] or DTL [18], to allow simple, low-cost wrappers to bus interfaces, and to allow backward compatibility with existing IPs. Our interface uses a request-response protocol that provides basic read and write operations. But it extends bus interfaces to fully exploit the power of our NoC [8, 20, 21]. For example, it offers connection-based communication where end-to-end flow control and time-related guarantees (e.g., bounded latency) can be requested.

The chapter is organized as follows. In the next two sections we compare NoC properties with those of off-chip networks and buses, respectively. In Section IV, we define the services that we offer in our network. Finally, we present our conclusions.

II. Networks Brought on Chip

Networks have been the subject of research for decades, both in the context of local and wide area networks (computer networks) [25], and as an interconnect for parallel machines [6]. Both are very much related to on-chip networks, and many of the results in those fields are also applicable on chip. However, NoC premises are different from off-chip networks, and, therefore, most of the network design choices must be re-evaluated.

NoCs differ from off-chip networks mainly in their constraints and synchronization. Typically, resource constraints are tighter on chip than off chip. Storage (i.e., memory) and computation resources are relatively more expensive, whereas the number of point-to-point links is larger on chip than off chip [7].

Storage is expensive, because general-purpose on-chip memory, such as RAMs, occupy a large area. Having the memory distributed in the network components in relatively small sizes is even worse, as the overhead area in the memory then becomes dominant.

For on-chip networks, computation too comes at a relatively high cost compared to off-chip networks. An off-chip network interface usually contains a dedicated processor to implement the part of the protocol stack, to relieve the host processor from the communication processing. Including a dedicated processor in a network interface may be not feasible on chip, as the size of the network interface will become comparable to or larger than the IP to be connected to the network. Moreover, running the protocol stack on the IP itself may also be not feasible, because often these IPs have one dedicated function only, and do not have the capabilities to run a network protocol stack. A cost-effective solution would be to have a dedicated-hardware implementation of the protocol stack.

The number of wires and pins to connect network components is an order of magnitude larger on chip than off chip [7]. If they are not used massively for other purposes than NoC communication, they allow wide point-to-point interconnects (e.g., 300-bit links) [7,15]. This is not possible off-chip, where links are relatively narrower: 8-16 bits.

On-chip wires are also relatively shorter than off chip [7], allowing a much tighter synchronization than off chip. This allows a reduction in the buffer space in the routers because the communication can be done at a smaller granularity. In the current semiconductor technologies, wires are also fast and reliable, which allows simpler link-layer protocols (e.g., no need for error correction, or retransmission). This also compensates for the lack of memory and computational resources.

In the rest of the section, we list five network issues that have a direct impact on the NoC cost: reliable communication, deadlock, data ordering, network flow control and buffering strategy, and time-related guarantees. For each of them, we discuss the differences and similarities for on- and off-chip networks.

Reliable communication. A consequence of the tight on-chip resource constraints is that the network components (i.e., routers and network interfaces) must be fairly simple to minimize computation and memory requirements. Luckily, on-chip wires currently provide a reliable communication medium, which can help to avoid the considerable overhead incurred by off-chip networks for providing reliable communication. Data integrity can be provided at low cost at the data link layer. However, data loss also depends on the network architecture. In most computer networks data is simply dropped if congestion occurs in the network [6,25]. On-chip, dropping data may lead to a too costly implementation of reliable communication. We show below that a network that does not drop data can be a much lower-cost solution, at the peril of introducing the possibility of deadlock.

Deadlock. Computer network topologies have generally an irregular (possibly dynamic) structure, which can introduce buffer cycles. In such topologies, packet dropping at the network nodes may be required to avoid deadlocks.

Deadlock can also be avoided without dropping data, for example by introducing constraints either in the topology or routing. Fat-tree topologies have already been considered for NoCs, where deadlock is avoided by bouncing back packets in the network in case of buffer overflow [9]. Tile-based approaches to system design [7, 15, 24] use mesh or torus network topologies, where deadlock can be avoided using, for example, a turn-model routing algorithm [6].

An alternative solution for deadlock in NoCs, which takes into consideration that modules connecting to the network are either masters (initiating requests and receiving responses), or slaves (receiving requests and sending back responses), is to maintain separate virtual networks (with separate buffers) for requests and responses [6].

Data ordering. In a network, data sent from a source to a destination may arrive out of order due to reordering in network nodes, following different routes, or retransmission after dropping. For off-chip networks out-of-order data delivery is typical. However, for NoCs where no data is dropped, data can be forced to follow the same path between a source and a destination (deterministic routing) with no reordering. This in-order data transportation requires less buffer space, and reordering modules are no longer necessary.

Network flow control and buffering strategy. Network flow control and buffering strategy have a direct impact on the memory utilization in the network. Wormhole routing requires only a flit buffer (per queue) in the router, whereas store-and-forward and virtual-cut-through routing require at least the buffer space to accommodate a packet [6]. Consequently, on chip, wormhole routing may be preferred over virtual-cut-through or store-and-forward routing. Similarly, input queuing may be a lower memory-cost alternative to virtual-output-queuing or output-queuing buffering strategies, because it has fewer queues. Dedicated (lower cost) fifo memory structures also enable on-chip usage of virtual-cut-through routing or virtual output queuing for a better performance [20]. However, using virtual-cut-through routing and virtual output queuing at

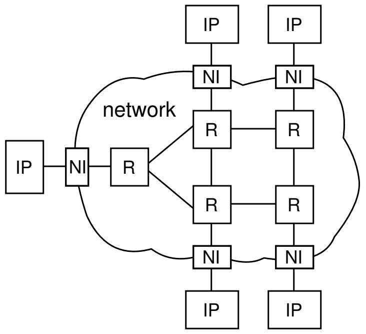

Figure 1. A network interconnect example

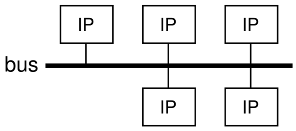

Figure 2. A bus interconnect example

the same time is still too costly [20].

Time-related guarantees. Off-chip networks typically use packet switching and offer best-effort services. Contention can occur at each network node, making latency guarantees very hard to offer. Throughput guarantees can still be offered using schemes such as rate-based switching [27] or deadline-based packet switching [19], but with high buffering costs.

An alternative to provide such time-related guarantees is to use time-division multiple access (TDMA) circuits, where every circuit is dedicated to a network connection. Circuits provide guarantees at a relatively low memory and computation cost. Network resource utilization is increased when the network architecture allows any left-over guaranteed bandwidth to be used by best-effort communication [10, 20, 21].

III. From buses to NoCs

Introducing networks (Figure 1) as on-chip interconnects radically changes the communication as compared to direct interconnects such as buses or switches (Figure 2). This is because of the multi-hop nature of a network, where communication modules are not directly connected, but separated by one or more network nodes. This is in contrast with the prevalent existing interconnects (i.e., buses) where modules are directly connected. The implications of this change reside in the arbitration (which must change from centralized to distributed), and in the communication properties (e.g., ordering, or flow control).

In this section, we list some of these topics, and outline the differences between NoCs and buses. We refer mainly to buses as direct interconnects, because currently they are the most used on-chip interconnect. Most of the bus characteristics also hold for other direct interconnects (e.g., switches [16]). Multilevel buses are a hybrid between buses and NoCs. For our purposes, depending on the functionality of the bridges, multilevel buses behave either like simple buses [2] or like NoCs.

Programming Model. The programming model of a bus typically consists of load and store operations which are implemented as a sequence of primitive bus transactions. Bus interfaces typically have dedicated groups of wires for command, address, write data, and read data [1, 12, 13, 17, 18, 26].

A bus is a resource shared by multiple IPs. Therefore, before using it, IPs must go through an arbitration phase, where they request access to the bus, and block until the bus is granted to them.

A bus transaction involves a request and possibly a response. Modules issuing requests are called masters, and those serving requests are called slaves. If there is a single arbitration for a request-response pair, the bus is called non-split. In this case, the bus remains allocated to the master of the transaction until the response is delivered, even when this takes a long time. Alternatively, in a split bus, the bus is released after the request to allow transactions from different masters to be initiated. However, a new arbitration must be performed for the response such that the slave can access the bus [11].

For both split and non-split buses, both communication parties have direct and immediate access to the status of the transaction. In contrast, network transactions are one-way transfers from an output buffer at the source to an input buffer at the destination that causes some action at the destination, the occurrence of which is not visible at the source [6]. The effects of a network transaction are observable only through additional transactions. A request-response type of operation is still possible, but requires at least two distinct network transactions. Thus, a bus-like transaction in a NoC will essentially be a split transaction.

Transaction Ordering. Traditionally, all transactions on a bus are ordered (cf. Peripheral VCI [26], AMBA [1], DTL [18], or CoreConnect PLB and OPB [12, 13]). This is possible at a low cost, because the interconnect, being a direct link between the communicating parties, does not reorder data. However, on a split bus, a total ordering of transactions on a single master may still cause performance penalties, when slaves respond at different speeds. To solve this problem, recent extensions to bus protocols allow transactions to be performed on connections. Ordering of transactions within a connection is still preserved, but between connections there are no ordering constraints (e.g., OCP [17], or Basic VCI [26]). A few of the bus protocols allow out-of-order responses per connection in their advanced modes (e.g., Advanced VCI [26]), but both requests and responses arrive at the destination in the same order as they were sent.

In a NoC, ordering becomes weaker. Global ordering can only be provided at a very high cost due to the conflict between the distributed nature of the networks, and the requirement of a centralized arbitration necessary for global ordering.

Even local ordering, between a source-destination pair, may be costly. Data may arrive out of order if it is transported over multiple routes. In such cases, to still achieve an in-order delivery, data must be labeled with sequence numbers and reordered at the destination before being delivered.

Atomic Chains of Transactions. An atomic chain of transactions is a sequence of transactions initiated by a single master that is executed on a single slave exclusively. That is, other masters are denied access to that slave, once the first transaction in the chain claimed it. This mechanism is widely used to implement synchronization mechanisms between master modules (e.g., semaphores).

On a bus, atomic operations can easily be implemented, as the central arbiter will either (a) lock the bus for exclusive use by the master requesting the atomic chain, or (b) not granting access to a locked slave. In the former case, the duration resources are locked is shorter because once a master has been granted access to a bus, it can quickly perform all the transactions in the chain (no arbitration delay is required for the subsequent transactions in the chain). Consequently, the locked slave and the bus can be opened up again in a short time. This approach is used in AMBA and CoreConnect.

In the latter case, the bus is not locked, and can still be used by other modules, however, at the price of a longer locking duration of the slave. This approached is used in VCI and OCP.

In a NoC, where the arbitration is distributed, masters do not know that a slave is locked. Therefore, transactions to a locked slave may still be initiated, even though the locked slave cannot accept them. Consequently, to prevent deadlock, these other transactions must be either dropped, or transactions in the atomic chain must be able to bypass them to be served. Moreover, the duration a module is kept locked is much longer in case of NoCs, because of the higher latency per transaction.

Deadlock. In buses, deadlock is generally not an issue. Deadlock can still occur at the application level (e.g., an atomic chain of transactions that locks the bus, but never unlocks it), but this is not caused by the interconnect itself.

In a network, deadlock becomes a more important issue, and special care has to be taken in the network design to avoid deadlock. Deadlock is mainly caused by cycles in the buffers. To avoid deadlock, either network nodes must drop packets when their buffers are filled, or routing must be cycle-free. In a NoC, we believe the latter is preferable, because of its lower cost in achieving reliable communication (see Section II).

A second cause of deadlock are atomic chains of transactions. The reason is that while a module is locked, the queues storing transactions may get filled with transactions outside the atomic transaction chain, blocking the access of the transaction in the chain to reach the locked module. If atomic transaction chains must be implemented (to be compatible with processors allowing this, such as MIPS), the network nodes should be able to filter the transactions in the atomic chain, or be allowed to drop those blocking them.

Media Arbitration. An important difference between buses and NoCs is in the medium arbitration scheme. In a bus, master modules request access to the interconnect, and the arbiter grants the access for the whole interconnect at once. Arbitration is *centralized* as there is only one arbiter component. It is also *global* as all the requests as well as the state of the interconnect are visible to the arbiter. Moreover, when a grant is given, the complete path from the source to the destination is exclusively reserved.

In a non-split bus, arbitration takes place once when a transaction is initiated. As a result, the bus is granted for both request and response. In a split bus, requests and responses are arbitrated separately.

In a NoC arbitration is also necessary, as it is a shared interconnect. However, in contrast to buses, the arbitration is *distributed*, because it is performed in every router, and is based only on *local* information. Arbitration of the communication resources (links, buffers) is performed incrementally as the request or response advances [20].

Destination Name and Routing. For a bus, the command, address, and data are broadcasted on the interconnect. They arrive at every destination, only one of which activates based on the broadcasted address, and executes the requested command. This is possible because all modules are directly connected to the same bus.

In a NoC, it is not feasible to broadcast information to all destinations, because it must be copied to all routers and network interfaces. This floods the network with data. The address is better decoded at the source to find a route to the destination module. A transaction address has, therefore, two parts: (a) a destination identifier, and (b) an internal address at the destination.

Latency. Transaction latency is caused by two factors: (a) the access time to the bus, which is the time until the bus is granted, and (b) the latency introduced by the interconnect to transfer the data.

For a bus, where the arbitration is centralized, the access time is proportional to the number of masters connected to the bus. The transfer latency itself typically is constant and relatively low, because the modules are linked directly. However, the speed of transfer is limited by the bus speed, which is relatively low.

In a NoC, arbitration is performed at each router for the following link. The access time per router is small. Both end-to-end access time and transport time increase proportionally to the number of hops between master and slave. However, network links are unidirectional and point to point and, hence, can run at higher frequencies than buses, thus lowering the latency.

From a latency prospective, using a bus or a network is a trade off between the number of modules connected to the interconnect (which affects access time), the speed of the interconnect, and the network topology.

Data Format. In most modern bus interfaces the data format is defined by separate wire groups for the transaction type, address, write data, read data, and return acknowledgments/errors (e.g., VCI, OCP, AMBA, DTL, or CoreConnect). This is used to pipeline transactions. For example, concurrently with sending the address of a read transaction, the data of a previous write transaction can be sent, and the data from an even earlier read transaction can be received. Moreover, having dedicated wire groups simplifies the transaction decoding; there is no need for a mechanism to select between different kinds of data sent over a common set of wires.

Inside a network, there is typically no distinction between different kinds of data. Data is treated uniformly and passed from one router to another. This is done to minimize the control overhead and buffering in routers. If separate wires would be used for each of the above-mentioned groups, separate routing, scheduling and queuing would be needed, and the cost of routers would increase proportionally.

In addition, in a network at each layer in the protocol stack, control information must be supplied together with the data (e.g., packet type, network address, or packet size). This control information is organized as an envelope around the data. That is, first a header is sent, followed by the actual data (payload), followed possibly by a trailer. Multiple such envelopes may be provided for the same data, each carrying the corresponding control information for each layer in the network protocol stack [6, 25].

Buffering and Flow Control. Buffering data of a master (output buffering) is used both for buses and NoCs to decouple computation from communication. However, for NoCs output buffering is also needed to marshal data, which consists of (a) (optionally) splitting the outgoing data in smaller packets which are transported by the network, and (b) adding control information for the network around the data (packet header). To avoid output buffer overflow the master must not initiate transactions that generate more data than the currently available space.

Similarly to output buffering, input buffering is also used to decouple computation from communication. In a NoC, input buffering is also required to unmarshal data.

In addition, flow control for input buffers differs for buses and NoCs. For buses, the source and destination are directly linked, and, destination can therefore signal directly to a source that it cannot accept data. This information can even be available to the arbiter such that the bus is not granted to a transaction trying to write to a full buffer.

In a NoC, however, the destination of a transaction cannot signal directly to a source that its input buffer is full. Consequently, transactions to a destination can be started, possibly from multiple sources, after the destination's input buffer has filled up. Several policies can be adopted when an input buffer is full. One policy is not to accept additional incoming transitions, and to store them in the network. However, this approach can easily lead to network congestion, as the data could be eventually stored all the way to the sources, blocking the links in between. Another policy is to accept incoming transactions at a full destination, and drop some data in the input buffer. Congestion is avoided but data is lost, and this is undesirable.

To avoid input buffer overflow connections can be used, together with end-to-end flow control. At connection set up between a master and one or more slaves, buffer space is allocated at the network interfaces of the slaves, and the network interface of the master is assigned credits reflecting the amount of buffer space at the slaves. The master can only send data when it has enough credits for the destination slave(s). The slaves grant credits to the master when they consume data.

IV. The Æthereal Approach

As described in the previous two sections, NoCs have different properties from both existing off-chip networks and existing on-chip interconnects. As a result, existing protocols and service interfaces cannot be adopted directly to NoCs, but must take the characteristics of NoCs into account. For example, a protocol such as TCP/IP assumes the network is lossy, and includes significant complexity to provide reliable communication. Therefore, TCP/IP is not suitable in a NoC where we assume data transfer reliability is already solved at a lower level. On the other hand, existing on-chip protocols such as VCI, OCP, AMBA, DTL, or CoreConnect are also not directly applicable. For example, they assume ordered transport of data: if two requests are initiated from the same master, they will arrive in the same order at the destination. This does not hold automatically for NoCs. Atomic chains of transactions and end-to-end flow control also need special attention in a NoC interface.

Our objectives when defining the services of our on-chip network (called Æthereal) are the following. First, the services abstract from the network internals as much as possible. This is a key ingredient in tackling the challenge of decoupling the computation from communication [14,23], which allows IPs (the computation part), and the interconnect (the communication part) to be designed independently from each other. As a consequence, our services are positioned at the transport layer in the ISO-OSI reference model [25], which is the first layer to be independent of the implementation of the network.

Second, we aim at a NoC interface as close as possible to a bus interface. NoCs can then be introduced non-disruptively: existing IPs, methodologies and tools can continue to be used with minor changes. As a consequence, we use a request-response interface similar to interfaces for split buses [1, 12, 13, 17, 18, 26].

Third, our interface extends traditional bus interfaces to fully exploit the power of NoCs. For example, we offer connection-based communication which does not only relax ordering constraints (as for buses), but also enables new communication properties, such as end-to-end flow control based on credits, or guaranteed throughput [8, 20, 21]. All these properties can be set for each connection individually.

A. The Æthereal Connection and Transaction Model

IPs interact with our network [8, 20, 21] at so-called network interfaces (NI). NIs provide NI ports (NIP) through which the communication services are accessed. As shown in Figure 3, a NI can have several NIPs to which one or more IPs (computation elements or memories, but not interconnection elements) can be connected. Similarly, an IP can be connected to more than one NI and NIP.

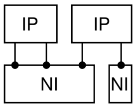

Figure 3. Examples of links between NIs and IPs.

Communication between NIPs is performed on *connections*. Connections are introduced to describe and identify communication with different properties, such as guaranteed throughput, bounded latency and jitter, ordered delivery, or flow control. For example, to distinguish and independently guarantee communication of 1Mbs and 25Mbs, two connections can be used. Two NIPs can be connected by multiple connections, possibly with different properties. Connections as defined here are similar to the concept of threads and connections from OCP and VCI. Where in OCP and VCI connections are used only to relax transaction ordering, we generalize from only the ordering property to include configuration of buffering and flow control, guaranteed throughput, and bounded latency per connection.

Æthereal connections must be *created* with the desired properties before being used. This may result in resource reservations inside the network (e.g., buffer space, or percentage of the link usage per time unit). If the requested resources are not available, the network will refuse the request. After usage, connections are *closed*, which leads to freeing the resources occupied by that connection.

To allow more flexibility in configuring connections and, hence, better resource allocation per connection, the outgoing and return parts of connections are configured independently. For example, a different amount of buffer space can be allocated in the NIPs at master and slaves, or different bandwidths can be reserved for requests and responses.

Depending on the requested services, the time to handle a connection (i.e., creating, closing, modifying services) can be short (e.g., creating/closing an unordered, lossy, best-effort connection) or significant (e.g., creating/closing a multicast guaranteed-throughput connection). Consequently, connections are assumed to be created, closed, or modified infrequently, coinciding e.g., with reconfiguration points, when the application requirements change.

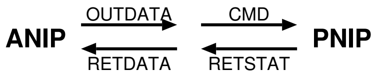

Figure 4. Transaction composition.

Communication takes place on connections using *transactions*, consisting of a request and possibly a response. The request encodes an operation (e.g., read, write, flush, test and set, nop) and possibly carries outgoing data (e.g., for write commands). The response returns data as a result of a command (e.g., read) and/or an acknowledgment.

Connections involve at least two NIPs. Transactions on a connection are always started at one and only one of the NIPs, called the connection's *active* NIP (ANIP). All the other NIPs of the connection are called *passive* NIP*s* (PNIP).

There can be multiple transactions active on a connection at a time, but more generally than for split buses. That is, transactions can be started at the ANIP of a connection while responses for earlier transactions are pending. If a connection has multiple slaves, multiple transactions can be initiated towards different slaves. Transactions are also pipelined between a single master-slave pair for both requests and responses. In principle, transactions can also be pipelined within a slave, if the slave allows this.

A transaction is composed from the following messages (see Figure 4):

- A *command* message (CMD) is sent by the ANIP, and describes the action to be executed at the slave connected to the PNIP. Examples of commands are read, write, test and set, and flush. Commands are the only messages that are compulsory in a transaction. For NIPs that allow only a single command with no parameters (e.g., fixed-size address-less write), we assume the command message still exists, even if it is implicit (i.e., not explicitly sent by the IP).
- An *out data* message (OUTDATA) is sent by the ANIP following a command that requires data to be executed (e.g., write, multicast, and test-and-set).
- A *return data* message (RETDATA) is sent by a PNIP as a consequence of a transaction execution that produces data (e.g., read, and test-and-set).
- A *completion acknowledgment* message (RETSTAT) is an optional

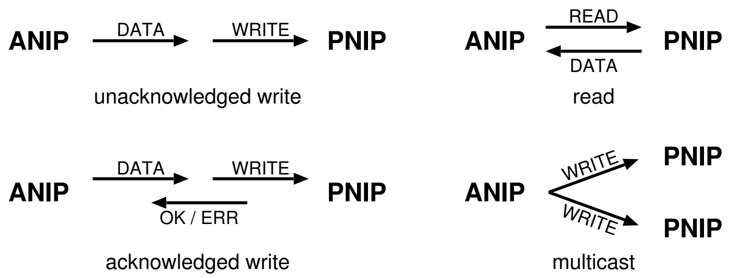

Figure 5. Transaction examples.

message which is returned by PNIP when a command has been completed. It may signal either a successful completion or an error. For transactions including both RETDATA and RETSTAT the two messages can be combined in a single message for efficiency. However, conceptually, they exist both: RETSTAT to signal the presence of data or an error, and RETDATA to carry the data. In bus-based interfaces RETDATA and RETSTAT typically exist as two separate signals [1, 12, 13, 17, 18, 26].

Messages composing a transaction are divided in *outgoing* messages, namely CMD and OUTDATA, and *response* messages, namely RETDATA, RETSTAT. Within a transaction, CMD precedes all other messages, and RETDATA precedes RETSTAT if present. These rules apply both between master and ANIP, and PNIP and slave. Examples of transactions are shown in Figure 5.

We classify connections as follows (see Figure 6):

- A *simple* connection is a connection between one ANIP and one PNIP.
- A *narrowcast* connection is a connection between one ANIP and one or more PNIPs, in which each transaction that the ANIP initiates is executed by exactly one PNIP. An example of the narrowcast connection is shown in Figure 7, where the ANIP performs transactions on an address space which is mapped on two memory modules. Depending on the transaction address, a transaction is executed on only one of these two memories.
- A *multicast* connection is a connection between one ANIP and one

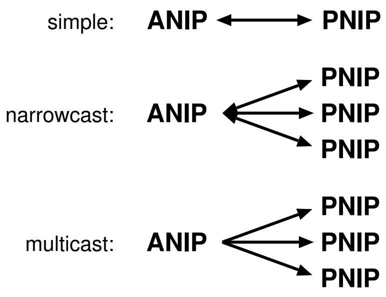

Figure 6. Connection types.

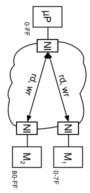

Figure 7. A narrowcast connection.

or more PNIPs, in which the sent messages are duplicated and each PNIP receives a copy of those messages. In a multicast connection no return messages are currently allowed, because of the large traffic they generate (i.e., one response per destination). It could also increase the complexity in the ANIP because individual responses from PNIPs must be merged into a single response for the ANIP. This requires buffer space and/or additional computation for the merging itself.

B. Connection Properties

In this section we elaborate on the features that can be configured for a connection: guaranteed message integrity, guaranteed transaction completion, various transaction orderings, guaranteed throughput, bounded latency and jitter, and connection flow control.

Data Integrity. Data integrity means that the content of messages is not changed (accidentally or not) during transport. We assume that data integrity is already solved at a lower layer in our network, namely at the link layer, because in current on-chip technologies data can be transported uncorrupted over links. Consequently, our network interface always guarantees that messages are delivered uncorrupted at the destination.

Transaction Completion. A transaction without a response (e.g., a posted write) is said to be complete when it has been executed by the slave. As there is no response message to the master, no guarantee regarding transaction completion can be given.

A transaction with a response (e.g., an acknowledged write) is said to be complete when a RETSTAT message is received from the ANIP[1]. The transaction may either (a) be executed successfully, in which case a success RETSTAT is returned, (b) fail in its execution at the slave, in which case an execution error RETSTAT is returned, or (c) fail because of buffer overflow in a connection with no flow control, in which case it reports an overflow error. We assume that when a slave accepts a CMD requesting a response, the slave always generates the response.

In our network, routers do not drop data [21]. Therefore, messages are always guaranteed to be delivered at the NI. For connections with flow control, also NIs do not drop data. Thus, message delivery and, thus, transaction completion to the IPs is guaranteed automatically in this case.

However, if there is no flow control, messages may be dropped at the network interface in case of buffer overflow (see the paragraph on end-to-end flow control below). All of CMD, OUTDATA, and RETDATA may be dropped at the NI. To guarantee transaction completion, RETSTAT is not allowed to be dropped. Consequently, in the ANIPs enough buffer space must be provided to accommodate RETSTAT messages for all outstanding transactions. This is enforced by bounding the number of outstanding transactions.

Transaction Ordering. Across different connections no ordering of transactions is defined at the transport layer.

There are several points in a connection where order of transactions can be observed (see Figure 8): (a) the order in which the master presents CMD messages to the ANIP, (b) the order in which the CMDs are delivered to the slave by the PNIP, (c) the order in which the slave presents the responses to the PNIP, and (d) the order the responses are delivered to the master by the ANIP. Note that not all of (b), (c), and (d) are always present. Moreover, there are no assumptions about the order in which the slaves execute

[1] Recall that when data is received as a response (RETDATA), a RETSTAT (possibly implicit) is also received to validate the data.

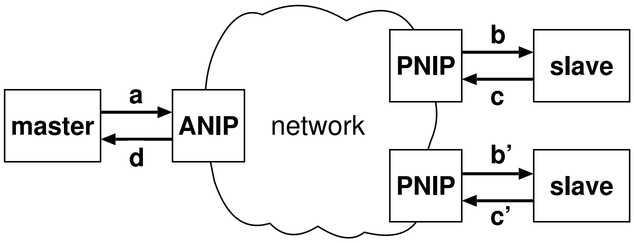

Figure 8. Message ordering is observable at a, b, c, and d.

transactions; we can only observe the order of the responses. We consider the order of the transaction execution by the slaves to be a system decision, and not a part of the interconnect protocol.

At both ANIP and PNIPs, outgoing messages belonging to different transactions on the same connection are allowed to be interleaved. For example, two write commands can be issued, and only afterwords their data. If the order of OUTDATA messages differs from the order of CMD messages, transaction identifiers must be introduced to associate OUTDATAs with their corresponding CMD.

Outgoing messages can be delivered by the PNIPs to the slaves (see Figure 8-b) as follows:

- *Unordered*, which imposes no order on the delivery of the outgoing messages of different transactions at the PNIPs.
- *Ordered locally*, where transactions must be delivered to each PNIP in the order they were sent (Figure 8-a), but no order is imposed across PNIPs. Locally-ordered delivery of the outgoing messages can be provided either by an ordered data transportation, or by reordering outgoing messages at the PNIP.
- *Ordered globally*, where transactions must be delivered in the order they were sent, across all PNIPs of the connection. Globally-ordered delivery of the outgoing part of transactions require a costly synchronization mechanism.

Transaction response messages can be delivered by the slaves to the PNIPs (see Figure 8-c) as follows:

- *Ordered*, when RETDATA and RETSTAT messages are returned in

the same order as the CMDs were delivered to the slave (Figure 8-b).
- *Unordered*, otherwise.

When responses are unordered, there has to be a mechanism to identify the transaction to which a response belongs. This is usually done using tags attached to messages for transaction identifications (similar to tags in VCI).

Response messages can be delivered by the ANIP to the master (see Figure 8-d) as follows:

- *Unordered*, which imposes no order on the delivery of responses. Here also, tags must be used to associate responses with their corresponding CMDs.
- *Ordered locally*, where RETDATA and RETSTAT messages of transactions for a single slave are delivered in the order the original CMDs were presented by the master to the ANIP (Figure 8-a). Note that there is no ordering imposed for transactions to different slaves within the same connection.
- *Globally ordered*, where all responses in a connection are delivered to the master in the same order as the original CMDs. When transactions are pipelined on a connection, then globally-ordered delivery of responses requires reordering at the ANIP.

All $3 \times 2 \times 3 = 18$ combinations between the above orderings are possible. Out of these, we define and offer the following two.

An *unordered* connection is a connection in which no ordering is assumed in any part of the transactions. As a result, the responses must be tagged to be able identify to which transaction they belong. Implementing unordered connections has low cost, however, they may be harder to use, and introduce the overhead of tagging.

An *ordered* connection is defined as a connection with *local* ordering for the outgoing messages from PNIPs to slaves (Figure 8-b), *ordered* responses at the PNIPs (Figure 8-c), and *global* ordering for responses at the ANIP (Figure 8-d). We choose local ordering for the outgoing part because the global ordering has a too high cost, and has few uses. The ordering of responses is selected to allow a simple programming model with no tagging. Global ordering at the ANIP is possible at a moderate cost, because all the reordering is done locally in the ANIP.

A user can emulate connections with global ordering of outgoing and return messages at the PNIPs using non-pipelined acknowledged transactions, at the cost of high latency.

Connection latency, throughput, and jitter. In our network, throughput can be reserved for connections in a time-division multiple access (TDMA) fashion, where bandwidth is split in fixed-size slots on a fixed time frame. Bandwidth, as well as bounds on latency and jitter, can be guaranteed when slots are reserved. They are all defined in multiples of the slots. throughput, latency and jitter can all be configured independently for the request and response parts of a connection.

Fully guaranteed-throughput connections (i.e., providing throughput guarantees on both request and return parts of the connection) can overbook resources in some cases. For example, when an ANIP opens a guaranteed-throughput read connection, it must reserve slots for the read command messages, and for the read data messages. The ratio between the two can be very large (e.g., 1:100), which leads either to a large number of slots, or bandwidth being wasted for the read command messages.

To resolve this problem, the request part of a connection can be best effort, while the response can have guaranteed throughput (or vice versa). For the example mentioned above, one can use best effort read messages, and guaranteed-throughput read-data messages. No global connection guarantees can be offered in this case, but the overall throughput can be higher and more stable than in the case of using only best-effort traffic.

Connection flow control. As mentioned earlier, our network guarantees that messages are delivered to the NI. Messages sent from one of the NIPs are not immediately visible at the other NIP, because of the multi-hop nature of networks. Consequently, handshakes over a network would allow only a single message be transmitted at a time. This limits the throughput on a connection and adds latency to transactions. To overcome this problem, and achieve a better network utilization, the messages must be pipelined. In this case, if the data is not consumed at the PNIP at the same rate it arrives, either flow control must be introduced to slow down the producer, or data may be lost because of limited buffer space at the consumer

NI.

We introduce end-to-end flow control at the level of connections, which requires buffer space to be associated with connections. End-to-end flow control ensures that messages are sent over the network only when there is enough space in the NIP's destination buffer to accommodate them.

End-to-end flow is optional (i.e., to be requested when connections are opened) and can be configured independently for the outgoing and return paths. When no flow control is provided, messages are dropped when buffers overflow. Multiple policies of dropping messages are possible, as in off-chip networks. Possible scenarios include: (a) the oldest message is dropped (milk policy), or (b) the newest message is dropped (wine policy) [25].

We opt for a credit-based flow control. Credits are associated with the empty buffer space at the receiver NI. The sender's credit is lowered as data is sent. When the PNIP delivers data to the slave, credits are granted to the sender. If the sender's credit is not sufficient to send some data, the NI at the sender stalls the sending.

C. Use Cases

To illustrate the need for differentiated services on connections, we consider in this section two examples of traffic. We describe the properties they would use over an Æthereal connection to meet their traffic requirements.

Video processing streams typically require a lossless, in-order video stream with guaranteed throughput, but possibly allow corrupted samples. An Æthereal connection for such a stream would require the necessary throughput, ordered transactions, and flow control. If the video stream is produced by the master, only write transactions are necessary. In this case, with a flow-controlled connection there is no need to also require transaction completion, because messages are never dropped, and the write command and its data are always delivered at the destination. Data integrity is always provided by our network, even though it may be not necessary in this case.

Another example is that of cache updates which require uncorrupted, lossless, low-latency data transfer, but ordering and guaranteed throughput are less important. In this case, a connection would not require any time related guarantees, because even though an average low latency is required, a guarantee on low latency is not critical. Low latency can be obtained even with a best-effort connection. The connection would also require flow control and guaranteed transaction completion to ensure lossless transactions. However, no ordering is necessary, because this is not important for cache updates, and allowing out of order transaction can reduce the response time.

V. Conclusions

In this chapter, we compare networks on chip (NoC) to off-chip networks (e.g., computer networks) and existing on-chip interconnects (e.g., buses). We show that NoCs have many similarities with off-chip networks. However, they also differ, especially in their resource constraints. For example on a chip, memory and computation resources are more expensive, while there are more wires. This makes NoC architectures different from off-chip networks, and requires rethinking of network services.

We also compare NoCs to existing on-chip interconnects, such as buses and switches. By directly connecting IP blocks, existing on-chip interconnects can offer tight coupling between masters and slaves, and global arbitration. In NoCs, masters and slaves are completely decoupled, and the arbitration is distributed over the network nodes. This make it harder to provide guarantees, such as bandwidth lower bounds, and transaction orderings.

We define a set of NoC services that abstract from the network details. Using these services in IP design decouples computation and communication. We use a request-response transaction model to be close to existing on-chip interconnect protocols. This eases the migration of current IPs to NoCs. To fully utilize the NoC capabilities, such as high bandwidth and transaction concurrency, our services provide connection-oriented communication. Connections can be configured independently with different properties. These properties include transaction completion, various transaction orderings, bandwidth lower bounds, latency and jitter upper bounds, and flow control.

Our NoC services are a prerequisite for service-based system design which makes applications independent of NoC implementations, makes designs more robust, and enables architecture-independent quality-of-service strategies.

References

1. ARM. *AMBA Specification. Rev. 2.0*, 1999.
2. ARM. *Multi-Layer AHB. Overview.*, 2001.
3. J. Bainbridge and S. Furber. CHAIN: A delay-insensitive chip area interconnect. *IEEE Micro*, 22(5), 2002.
4. L. Benini and G. De Micheli. Powering networks on chips. In *ISSS*, 2001.
5. L. Benini and G. De Micheli. Networks on chips: A new SoC paradigm. *IEEE Computer*, 35(1):70–80, 2002.
6. D. J. Culler, J. P. Singh, and A. Gupta. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann Publishers, 1999.
7. W. J. Dally and B. Towles. Route packets, not wires: On-chip interconnection networks. In *DAC*, 2001.
8. K. Goossens, J. van Meerbergen, A. Peeters, and P. Wielage. Networks on silicon: Combining best-effort and guaranteed services. In *DATE*, 2002.
9. P. Guerrier and A. Greiner. A generic architecture for on-chip packet-switched interconnections. In *DATE*, 2000.
10. P. J. Having. *Mobile Multimedia Systems*. PhD thesis, University of Twente, 2000.
11. J. Hennessy and D. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 1995.
12. IBM. *32-bit On-Chip Peripheral Bus. Rev. 2.1*, 2001.
13. IBM. *32-bit Processor Local Bus. Rev. 2.9*, 2001.
14. K. Keutzer, S. Malik, A. R. Newton, J. M. Rabaey, and A. Sangiovanni-Vincentelli. System-level design: Orthogonalization of concerns and platform-based design. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 19(12):1523–1543, 2000.

15. S. Kumar, A. Jantsch, J.-P. Soininen, M. Forsell, M. Millberg, J. Öberg, Tiensyrjä, and A. Hemani. A network on chip architecture and design methodology. In *ISVLSI*, 2002.
16. J. A. Leijten, J. L. van Meerbergen, A. H. Timmer, and J. A. Jess. Prophid, a data-driven multi-processor architecture for high-performance DSP. In *ED&TC*, 1997.
17. OCP International Partnership. *Open Core Protocol Specification*, 2001.
18. Philips. *DTL Protocol Specification. Rev. 2.1*, 2001.
19. J. Rexford. *Tailoring Router Architectures to Performance Requirements in Cut-Through Networks*. PhD thesis, Univ. Michigan, 1999.
20. E. Rijpkema, K. Goossens, A. Rădulescu, J. Dielissen, J. van Meerbergen, P. Wielage, and E. Waterlander. Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip. In *DATE*, 2003.
21. E. Rijpkema, K. Goossens, and P. Wielage. A router architecture for networks on silicon. In *PROGRESS*, Oct. 2001.
22. M. T. Rose. *The Open Book: A Practical Perspective on OSI*. Prentice Hall, 1990.
23. M. Sgori, M. Sheets, K. Keutzer, S. Malik, J. Rabaey, and A. Sangiovanni-Vincentelli. Addressing the system-on-a-chip interconnect woes through communication-based design. In *DAC*, 2001.
24. P. Stravers and J. Hoogerbrugge. Homogeneous multiprocessing and the future of silicon design paradigms. In *VLSI-TSA*, 2001.
25. A. S. Tanenbaum. *Computer Networks*. Prentice Hall, 1996.
26. VSI Alliance. *Virtual Component Interface Standard*, 2000.
27. H. Zhang. Service disciplines for guaranteed performance service in packet-switching networks. *Proc. of the IEEE*, 83(10):1374–1396, 1995.