The *Nostrum* Backbone - a Communication Protocol Stack for Networks on Chip

Mikael Millberg, Erland Nilsson, Rikard Thid, Shashi Kumar and Axel Jantsch

Laboratory of Electronic & Computer Systems,
Royal Institute of Technology (KTH),
LECS/IMIT/KTH-Electrum, Electrum 229, SE-164 40 Kista, Sweden
Email: {micke, erlandn, thid, shashi, axel}@imit.kth.se

Abstract

We propose a communication protocol stack to be used in Nostrum, our Network on Chip (NoC) architecture. In order to aid the designer in the selection process of what parts of protocols, and their respective facilities, to include, a layered approach to communication is taken. A nomenclature for describing the individual layers' interfaces and service definitions of the layers in the protocol stack is suggested and used. The concept includes support for best effort traffic packet delivery as well as for guaranteed bandwidth traffic using virtual circuits. Furthermore an application to NoC adapter is defined as part of the Resource to Network Interface, used to communicate between the Nostrum protocol stack and the application. An industrial example has been implemented and simulated and the results justifies the suggested layered approach.

1 Introduction

Current core based *System-on-Chip (SoC)* methodologies do not offer the required amount of reuse to enable the system designer to meet the time to market constraint. A future SoC methodology should have potential of not only reusing cores but also reusing the interconnection and communication infrastructure among cores.

The need to organise a large number of cores on a chip using a standard interconnection infrastructure has been realised for quite some time. This has led to proposals for platform based designs using standardised interfaces, e.g. the VSI initiative [1]. Platforms usually contain bus based interconnection infrastructures, where a designer can create a new system by configuring and programming the cores connected to the busses. Due to the need for a systematic approach for designing on-chip communication communication centric design methodologies have been proposed in [2, 3] where interconnection and communication among cores for a SoC will captivate the major portion of the design and test effort.

As recognised in [4], bus based platforms suffer from limited scalability and poor performance for large systems. This has led to proposals for building regular packet switched networks on chip as proposed in [5, 6, 7]. These *Network-on-Chips (NoCs)* are the network based communication solution for SoCs. They allow reuse of the communication infrastructure across many products thus reducing design-and-test effort and time-to-market.

In order to enhance reusability and to ease programmability, all NoC proposals recommend standardised and layered protocols for communication among cores. While some of these papers discuss the protocol layering on a conceptual and abstract level [2, 6, 8] others have only elaborated and implemented the lower protocol layers [5, 12].

In this paper, a concrete instance of a communication protocol stack for Nostrum, our NoC architecture, from physical to transport layer is proposed. One important aspect of our approach is the connection to geometry and implementation, which will help us to find efficient solutions.

2 *Nostrum*

We have developed a concept that we call *Nostrum* [13] that is used for defining a concrete architecture – the *Nostrum Mesh Architecture*. The communication infrastructure used within the concept is called the *Nostrum Backbone*.

2.1. The *Nostrum* Concept

Nostrum is our concept of network based communication for 'System on Chip's (SoCs). *Nostrum* mixes traditional mapping of applications to hardware with the use of the communication infrastructure offered by Network-on-chip (NoCs). Within *Nostrum*, the '*System*' in SoC can be seen as a system of applications consisting of one or more processes. In order to let processes communicate, *Nostrum* offers a packet switched communication platform that can be reused for a large number of SoCs.

To make the packet switched communication practical for on-chip communication, the protocols used in traditional computer networks cannot be used directly; they need to be simplified so that the implementation cost as well as speed/throughput performance is acceptable. These simplifications are made from a functional point of view and only a limited set of functions are realised.

2.2. The *Nostrum* Backbone

The purpose of the backbone is to provide a reliable communication infrastructure, where the designer can explore and chose from a set of implementations with different levels of reliability, complexity of service etc.

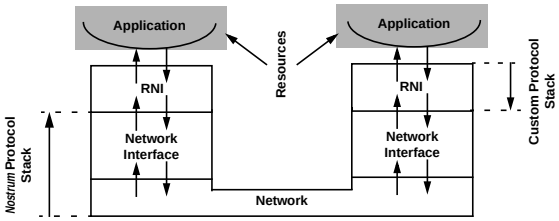

Figure 1. The Application/RNI/NI

In order to make the resources communicate over the network, resources are equipped with a *Network Interface (NI)*. The NI provides a standard set of services that can be utilised by the *Resource Network Interface (RNI)* or by the resource directly. The role of the RNI is to act as glue (or adaptor) between the resource's internal communication infrastructure and the standard set of services of the NI. Dependent on the functionality requested from the *Nostrum backbone*, the *Nostrum* protocol stack can be more or less shallow, but the depth of the custom protocol stack, which may include the RNI, is not specified within *Nostrum*.

The backbone can be used for both best-effort traffic using single-message passing between resources where switching decisions are made locally in the switches on a dynamic/non-deterministic basis for every individual datagram routed through the network, as well as for guaranteed bandwidth traffic using virtual circuits.

2.3. The *Nostrum* Mesh Architecture

The NoC *Nostrum* Mesh Architecture (described in [14]) is a concrete instance of the *Nostrum* concept and consists of *Resources (R)* and *Switches (S)* organised, logically and physically in a structure where each switch is connected to

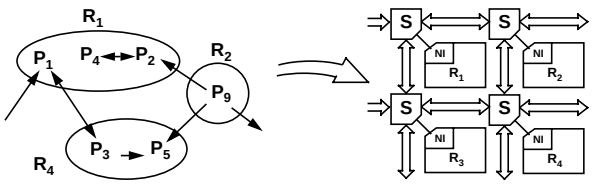

Figure 2. *Nostrum* Process to Resource mapping

its switch neighbours and to its resource as depicted in Figure 2. From an implementation point of view, the resources (Processor cores, DSPs, Memories, I/O etc.) are the realisation of the *Processes (P)*. A resource can host one or many processes, potentially the processes can belong to one or several different applications. However, the *Nostrum* concept is not inherently dependent of the mesh topology, other possibilities might include folded torus, fat-trees [9] etc.

The reason why the mesh topology was chosen stems from reasons of three types:

First, higher order dimension topologies are hard to implement. As analysed in [10], low dimension topologies are favoured when wiring and interconnects carry a significant cost, there is a high bandwidth between switches, and the delay caused by switches is comparable to the inter-switch delay. The torus topology was rejected in favour of a mesh since the folded torus has longer inter-switch delays.

Second, there is no real need for higher order dimension topologies, assuming that all applications we have in mind, e.g. telecom equipment, terminals, multi-media devices, consumer electronics etc. exhibit a high degree of locality in the communication pattern. This in stark contrast to the traditional parallel computers; designed to minimise latency for arbitrary communication.

Third, the mesh inhibits some desirable properties of its own, such as a very simple addressing scheme and multiple source-destination routes, which give robustness against network disturbances.

3 The Layered Communication Approach

In order to make different processes communicate a standard need to be defined for the format of communication. The requirements are quite diverse; the process communication mechanism needs to be able to deal with different data formats, different priorities, the interaction with the underlying network etc.

In order to implement this functionality a vertical set of layers has been defined, with each layer providing the layer above and below with a set of services. This layered approach to communication to great extent enhances the possibilities of performing simulations and implementations of the different layers in a multitude of design/model/specification languages. It also alleviates changes in the protocol stack.

3.1. Terminology

In order to describe the functionality of the different layers we use a terminology where similar functionalities are grouped into layers and a service is defined as an agreed functionality, which the particular layer has to provide. The services a layer offers could further be divided into functions implemented in that particular layer. The functions within a layer are collected into groups called

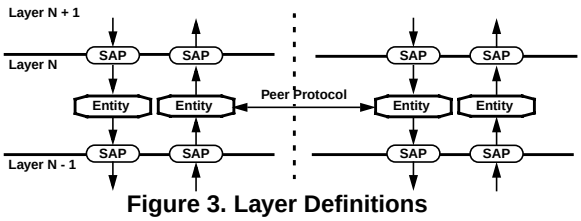

Figure 3. Layer Definitions

entities. These entities, within the same layer communicate with their peers using one or more peer protocols as seen in Figure 3. These units of information used to implement the protocol are called Protocol Data Units (PDUs), i.e. the PDU is the agreed data format, used by peers. In order to provide these services, an interface to the upper and lower layer has to be defined. These interfaces are called Service Access Points (SAPs).

4 The *Nostrum* Protocol Stack

As a starting point for the layers employed in the *Nostrum* backbone protocol stack, the abstraction levels of the OSI reference model is used as a basis. However, these layers only exist as a conceptual aid in the design process. Once the design is set, the layers might be collapsed at a logic level in order to enhance the possibility of hardware implementation optimisations. Within the concept of the *Nostrum* backbone resides the three "compulsory" layers; Network, Data Link, and Physical Layer. These three layers provide the service of delivering packets with a Destination Process Identifier (DPID) as the destination address.

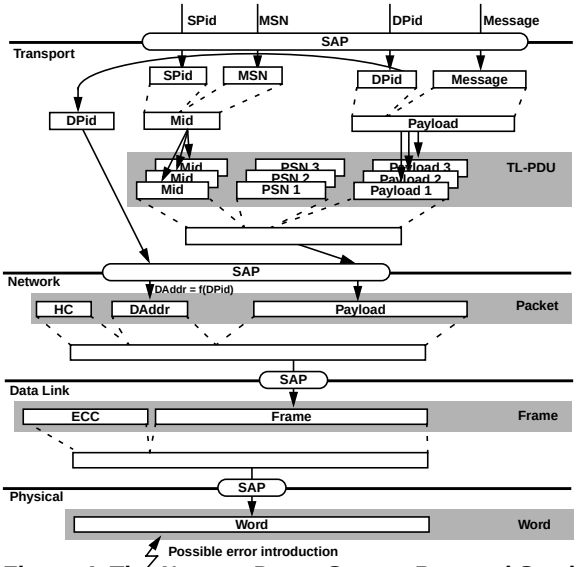

Figure 4. The *Nostrum* Down-Stream Protocol Stack

4.1. Physical Layer

The lowest layer is the Physical Layer with the purpose of move a word from the output of a switch to the corresponding input of the next. From a simulation point of view, this gives the possibility of error introduction.

4.2. Data Link Layer

The Data Link Layer (LL) provides the reliable transfer of information across the physical link. The LL is responsible for the transmission of frames with the necessary synchronisation, error control, and flow control.

4.3. Network Layer

The Network Layer's (NL) responsibility is to provide the transport of packets from the transmitting resource's NI through the network to the receiving resource ditto. The NL has the responsibility of packet routing. The NL also has the responsibility of managing and mapping the physical/logical addresses to the process identifiers associated with the processes in the different resources. This management involves the mapping/re-mapping of the process-resource pair, either at system set-up or during run-time.

At the NL two different services can be requested. The first service is the datagram service delivery; any packet sent to the NL will get routed dynamically through the network using a variant of deflective routing [11, 14].

The second service is the virtual circuit switching service. On request, a virtual circuit between the transmitter and the receiver is set up. This virtual circuit is implemented as a modified TDMA where the circuit can allocate fractions of the total bandwidth.

4.4. Transport Layer

The Transport Layer (TL) handles the establishment of communication. In the case of virtual circuit it issues an establishment of a connection and handles the transfer of data, i.e. it ensures a virtual point-to-point connection. It also provides f traffic control, i.e. the load of the network is detected and overload of the network is hence avoided.

5 The *Nostrum* Simulator

A simulator has been developed in order to prove the *Nostrum* concept. The simulator implements the different layers of the protocol stack "independently", i.e. different layers are implemented as separate plug-ins. The choice of the plug-ins is a part of the design process. Within one design, several different plug-ins can be used for implementing the same layer dependent on the communication requirements. To test the *Nostrum* simulator a relatively large industrial example in the form of a distributed DSP core was employed. The purpose of a real-life example is twofold; it justifies *Nostrum* as a concept and it justifies the use of the simulator as a platform to demonstrate the working of our ideas.In the current simulation set-up only the Application and the Network Layer (NL) were fully simulated. Due to

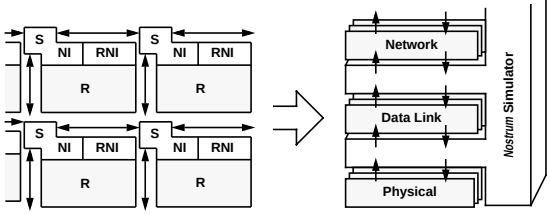

Figure 5. The Simulated *Nostrum* Protocol Stack

this, packets are passed directly to the NL, ignoring the intermediate layers. In a full simulation, the diversity in message sizes would force us to split and merge the messages in the TL.

5.1. The Distributed DSP Core Model

Ericsson Radio Systems provided a typical application from the real world. It is a relatively large industrial example in the form of a distributed DSP core.

The application has 12 serial inputs and 12 serial outputs. The input streams are de-serialised, characterised and sent to the appropriate *Queue*. After these respective data streams have been processed by FPGAs and DSPs they are re-serialised and sent out on one of the outgoing channels. The *Control Processor*, which loads all the system components with data at start-up hosts the monitor and control facilities, after start-up the traffic that it generates is negligible. The characteristics of the messages between the FPGAs and DSPs are shown in the Table 1.

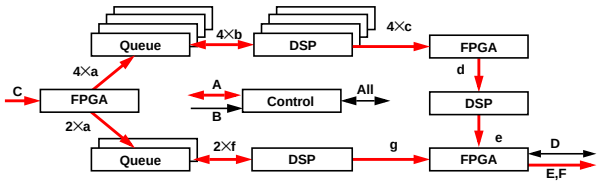

Figure 6. The Distributed DSP Core Example

Type	Bit- width	Rate (MHz)	Class
a	128	32	Continuous
b, c	16	32	Burst
d	64	32	Burst
e	32	16	Burst
f	16	8	Burst
g	16	4	Burst

Table 1. Channel Characteristics

5.2. Conclusions drawn from Simulation

The simulations performed show that:

- The protocol stack defined is useful for the provided application example.
- The simulator correctly implements the protocol stack.
- We can simulate fairly complex applications on a very high abstraction level, which only captures the traffic pattern between processes.

In addition to experiment with the protocol stack and to validate the simulator we could also draw several conclusions about requirements imposed by applications on the *Nostrum*. The two most important are: First, the traffic is indeed highly local provided that we have a sensible mapping of tasks into resources. Second, multi-cast messages have the potential of significantly decrease the communication load and improve performance, because, data streams are often fed to several consumers which process data in parallel. Since these streams may constitute high traffic loads it is beneficial to split the streams into multiple copies as close to the consumers as possible.

6 Conclusions

A standard protocol stack for a Network-on-Chip platform allows the separate, independent development and validation of resources, communication network and applications as long as they comply with the defined protocols. This clear separation enables systematic reuse of resources, communication infrastructure and application features.

We have defined such a protocol stack within the concept of *Nostrum*. The protocol stack is defined from the physical to the transport layer and, based on it, the layered *Nostrum* simulator is developed. The simulator allows us to experiment with different protocol variants because individual layers can be replaced without affecting other parts. Furthermore, we have modelled and simulated a real application on a very high level of abstraction only capturing the traffic pattern between tasks and resources. This allows us to analyse application requirements in order to optimise the protocol stack and the *Nostrum* architecture.

REFERENCES

[1] Virtual Socket Interface Alliance. http://www.vsi.org

[2] L. Benini and G. DeMicheli. Networks on chip: A new SoC paradigm. IEEE Computer, 35(1): p. 70 ff., Jan. 2002.

[3] P. Wielage and K. Goossens. Networks on Silicon: Blessing or Nightmare?. In Proc. of Euromicro Symposium on Digital System Design. Architectures, Methods and Tools, 2002, p 196-200

[4] P. Guerrier and A. Greiner. A generic architecture for on-chip packet-switched interconnections. In Proc. of DATE 2000, March 2000.

[5] W. J. Dally and B. Towles. Route packets, not wires: On-chip interconnection networks. In Proc. of DAC 2001, June 2001.

[6] M. Sgroi et al. Addressing the system-on-a-chip interconnect woes through communication-based design. Proc. of DAC 2001, June 2001.

[7] S. Kumar et al. A network on chip architecture and design methodology. IEEE Computer Society. Annual Symposium on VLSI, April 2002.

[8] K. Goossens et al., Networks on Silicon: Combining Best-Effort and Guaranteed Services, DATE 2002, March 2002.

[9] C. E. Leiserson. Fat Trees: Univ. Networks for Hardware Efficient Supercomputing. IEEE Computer, p 892 ff. vol. c-34, No 10, Oct. 1985.

[10] D. E. Culler and J. P. Singh, “Parallel Computer Architecture - a Hardware Software Approach”, 1999, Morgan Kaufmann Publishers, Inc., ISBN 1-55860-343-3

[11] Feige, U.; Raghavan, P. Exact analysis of hot-potato routing. In Proc. of Foundations of Computer Science, p. 553 -562, 1992

[12] D. Wingard, “MicroNetwork-Based Integration of SoCs”, In Proc. of DAC 2001, June 2001.

[13] M. Millberg, The Nostrum protocol stack and suggested services provided by the Nostrum backbone, Technical Report TRITA-IMIT-LECS R 02:01, LECS, Dept. of IMIT, KTH, Stockholm, Sweden, 2003

[14] Erland Nilsson et al., Load distribution with the proximity congestion awareness in a NoC. DATE 2003, Munich, Germany, March 2003.