

Matteo Dall'Osso[†], Gianluca Biccari[†], Luca Giovannini[†], Davide Bertozzi[†] and Luca Benini[†]

[†] University of Bologna, DEIS, Viale Risorgimento, 2 Bologna (Italy)

{mdallosso, gbiccari, lgiovannini, dbertozzi, lbenini@deis.unibo.it}

Abstract

The growing complexity of customizable embedded multi-processor architectures for digital media processing will soon require highly scalable network-on-chip based communication infrastructures. In this paper, we propose *xpipes*, a scalable and high-performance NoC architecture for multi-processor SoCs, consisting of soft macros that can be turned into instance-specific network components at instantiation time. The flexibility of its components allows our NoC to support both homogeneous and heterogeneous architectures. The interface with IP cores at the periphery of the network is standardized (OCP-based). Links can be pipelined with a flexible number of stages to decouple data introduction speed from worst-case link delay. Switches are lightweight and support reliable communication for arbitrary link pipeline depths (latency insensitive operation). *xpipes* has been described in synthesizable SystemC, at the cycle-accurate and signal-accurate level.

1 Introduction

To meet communication requirements on future SoCs, two critical challenges must be addressed, namely, system interconnect *delay* and *scalability*.

Technology scaling projections show that for 50 nm technology, crossing a chip with highly optimized interconnects will take between 6 to 10 clock cycles, therefore only a fraction of the chip area (between 0.4 and 1.4%) will be reachable in one clock cycle [5]. This effect will pose serious limitations to the maximum achievable SoC operating frequency. *Latency-insensitive* design approaches have been proposed as a workaround for the interconnect delay problem [4, 6]. In latency-insensitive designs, wires are divided into segments with propagation time compatible with clock cycle budgets, and signals propagate along the segments in a pipelined fashion. The main advantage is that clock speed (and link throughput) can be increased past the wire delay limit, but communication interfaces are required to provide buffering and synchronization, i.e., to become *insensitive* to link latency.

In this paper, we aim at addressing both delay and scalability challenges. The main contribution of this work is to introduce a highly parameterizable (i.e., soft), flexible, scalable and latency-insensitive NoC design (codenamed *xpipes*¹) targeting both homogeneous and heterogeneous MP-SoC architectures. The high flexibility of our imple-

mentation is due to the possibility of setting network parameters at instantiation time, thus obtaining an instance-specific design from a general-purpose parameterized one. NoC parameters have been selected to provide a wide design space and include: topology, link width, number of virtual channels, number of switch I/Os. *xpipes* soft macros include both switch and network interface (which provides a standardized, OCP-based interface with network nodes), featuring a highly optimized architecture for performance and low area overhead.

Network links can be pipelined with a flexible number of stages, thereby decoupling link throughput from its physical length. Switch and network interface are latency insensitive, in that they ensure correct operation for arbitrary link pipeline depth. As a result, data flows through the network at rates which are not bound by the worst-case delay of its longest link. *xpipes* also features latency-insensitive error control schemes which ensure robustness against communication errors.

2 Previous work

The need to progressively replace on-chip buses with micro-networks was extensively discussed in [7]. The Sonics MicroNetwork [8] is an example of evolutionary solution, which moves from a physical implementation as a shared bus, and proposes generalizations to support higher bandwidth (such as partial and full crossbars). However, in the long term more radical solutions are required.

In this direction, Dally and Lacy sketch the architecture of a VLSI multi-computer using 2009 technology [9], drawing upon past experiences in designing parallel computers and reconfigurable architectures [1, 2, 3].

Most proposed NoC platforms are packet switched and exhibit regular structure. The NOSTRUM network described in [10] takes this approach. The Scalable Programmable Integrated Network (SPIN) described in [11] is another regular, fat-tree-based network architecture.

The need to map communication requirements of heterogeneous IP blocks sometimes leads to the adoption of irregular topologies, due to the fact that each block can be optimized for a specific application (e.g., video or audio processing). Supporting heterogeneous architectures requires a major design effort and leads to coarser-granularity control of physical parameters [12]. The *æthereal* NoC design framework aims at providing a complete infrastructure for developing heterogeneous NoCs with end-to-end quality of service guarantees [13]. One of the earliest contributions in this area is the Maia het-

¹To be read as "crosspipes", a short-hand for "crossing pipelines"

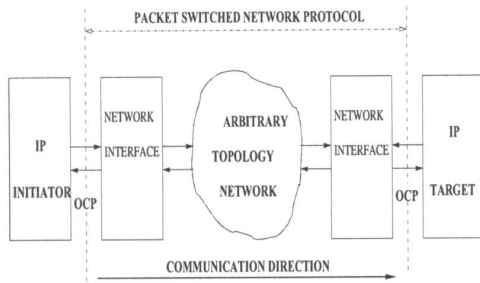


Figure 1: Block diagram of the network architecture.

erogeneous signal processing architecture, based on a hierarchical mesh network [14]. Maia's interconnect is fully instance-specific and communication is based on circuit switching, as opposed to packet switching.

The full support for heterogeneous architectures while still retaining instantiation time flexibility, packet switching and high performance are the distinctive characteristics of our NoC implementation.

3 $\times$ pipes architecture

Our network design methodology is divided into two fundamental steps. At first, a parameterizable SystemC description of all network components is provided, which serves as a general purpose design template. Then, at instantiation time the individual components are connected and configured by specifying the instance-specific parameters. This approach makes $\times$ pipes designs flexible, modular and fully scalable.

3.1 Architecture overview

A pictorial summary of our network implementation is shown in Fig. 1. Details about the communication protocol within the network are completely hidden to the IP blocks, which can be easily plugged into the architecture provided they are compliant with a standard end-to-end *OCP* (Open Core Protocol) [15] protocol. *OCP* exhibits a high degree of configurability to adapt to the core's functionality and it separates requests from responses, thus supporting multiple outstanding requests and therefore pipelining of transfers.

The *network interface* connects the IP block to the SoC interconnect. Its main objectives are data packetization and communication protocol conversion (from end-to-end to network protocol). Each packet is broken into message flow control units or *flits*. In our NoC design, a single flit can be pushed into the network in one clock cycle.

Overall, the network infrastructure consists of a collection of mutually connected switches in arbitrary topology (mesh, binary tree, butterfly, torus, etc.). Latency insensitive operation of the switches has been guaranteed, as will be described in next subsections.

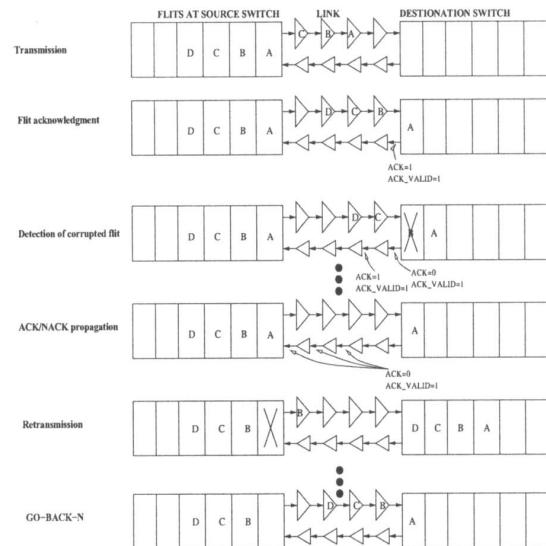


Figure 2: Pipelined switch-to-switch link model and its effect on retransmission of corrupted flits.

The NoC backbone relies on a wormhole switching technique [1], that aims at minimizing area- and power-hungry buffering resources at the switches. This goal is pursued also by using forward flow control: a flit is transmitted to the next switch along the path only when available storage resources are available. Our switch module supports multiple virtual channels multiplexed across the same physical link, so to improve flit latency and network throughput. A physical link is assigned to the different virtual channels on a flit-by-flit basis.

We adopted a static routing algorithm called *street sign routing*. Routes are derived by the network interface by accessing a look-up table based on the destination address. Such routes consist of direction bits read by each switch and indicating the output port of the switch which flits belonging to a certain packet have to be directed to. This routing algorithm allows a lightweight switch implementation with respect to the use of dynamic techniques.

3.2 Point-to-point link model

Our switch-to-switch link is subdivided into basic segments whose length guarantees that the desired clock frequency (i.e. the maximum speed provided by a certain technology) can be used. According to the link length, a certain number of clock cycles is needed by a flit to cross the interconnect. If network switches are designed in such a way that their functionality correctness depends on the flit arriving order and not on their timing, input links of the switches can be different and of any length. These design choices are at the basis of latency insensitive operation of the NoC and allow the construction of an arbitrary network topology and hence support for heterogeneous architectures.

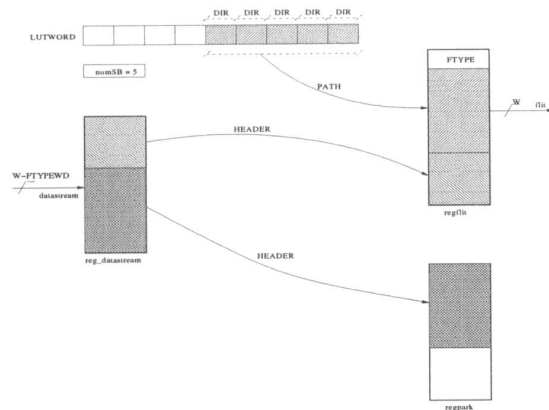


Figure 3: Mechanism for building header flits at the network interface.

Fig. 2 illustrates the link model, which is equivalent to a pipelined shift register. Pipelining has been used both for data and control lines. The figure illustrates how this model affects the retransmission of a corrupted flit between two successive switches in our implementation. Multiple outstanding flits propagate across the link during the same clock cycle. The retransmission strategy we implemented is GO-BACK- N , to keep the switch complexity as low as possible.

3.3 Network interface

The network interface (NI) for IP blocks that initiate communication needs to turn OCP-compliant transactions into packets to be transmitted across the network.

The NI has to build the packet header, which has to be spread over a variable number of flits depending on the length of the path to the destination node. The routing information is read by accessing a look-up table based on the destination node.

Based on input data *datastream* from the master IP and the look-up table word *LUTWORD*, header flits are formed as illustrated in Fig. 3. Let us assume that a packet requires $numSB = 5$ hops to get to destination, and that the direction to be taken at each switch is expressed by *DIR*. By concatenating the flit type *FTYPE* field with path information, the first flit is built. If there is some space left in the flit, it is filled with header information derived from the input *datastream*. The unused part of *datastream* is stored in a *regpark* register, so that a new *datastream* can be read. The following header and/or payload flits will be formed by combining data stored in *regpark* and *reg_datastream*. No partially filled flits are transmitted to make transmissions more efficient.

The response phase mainly consists of buffering resources to synchronize network's requests with the IP consuming rate. When a read transaction is initiated by the master IP, a *start_receive_response* signal is asserted, which

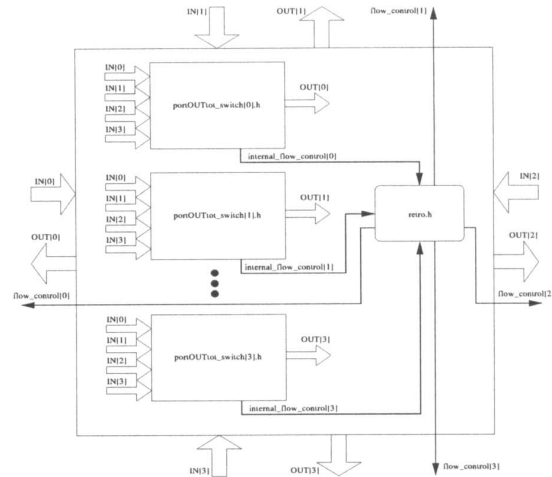


Figure 4: Overview of switch architecture. The number of inputs and outputs are design parameters.

triggers the waiting phase of the receive module for the requested data. As a consequence, NI supports only one outstanding read operation to keep interface complexity low, although an indefinite number of write transactions can be carried out after an outstanding read is initiated.

3.4 Switch

Latency insensitive operation of the switch involves virtual channel registers able to contain at least $2N + M$ flits, where N is the link length (expressed as number of basic repetition stages) and M is a switch architecture related contribution (12 cycles in our design). The reason is that each transmitted flit has to be acknowledged before being discarded from the buffer. Before an ACK is received, the flit has to travel across the link (N cycles), an ACK/NACK decision has to be taken at the destination switch (a portion of M cycles), the ACK/NACK signal has to be propagated back (N cycles) and recognized by the source switch (remaining portion of M cycles). During this time, other $2N + M$ flits are transmitted but not yet ACKed.

Switch design parameters are: number of inputs and outputs, flit width, number of virtual channels, length of switch-to-switch links, size of output registers. We chose output buffering for $\times pipes$ switches, and the resulting architecture is reported in Fig. 4. It consists of a replication of the same output module, accepting all input ports as its own input. Flow control signals generated by each output block are directed to a centralized module, that takes care of generating proper ACKs or NACKs for the incoming flits from the different input ports.

Each output module is deeply pipelined (7 pipeline stages) so to maximize the operating clock frequency of the switch. The detailed architecture is illustrated in Fig. 5. We observe that the impact of error detection on switch

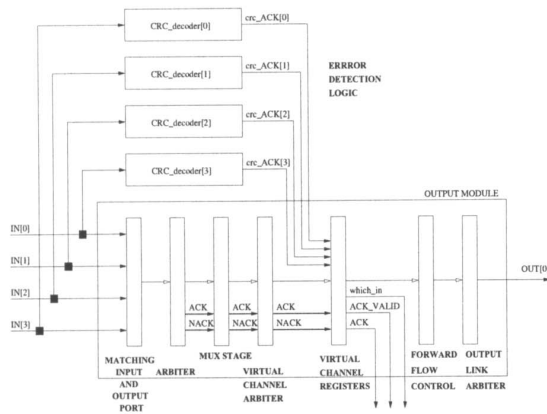


Figure 5: Architecture of a switch output module.

performance is hidden, in that CRC decoders work in parallel with respect to switch operation, although they have to complete within 4 clock cycles.

The first pipeline stage checks the header of incoming packets on different input ports to determine whether those packets have to be routed through the output port under consideration. Only matching packets are forwarded to the second stage, which resolves contention based on a round robin policy. Arbitration is carried out against receipt of tail flits, so that all flits of a packet can be propagated without contention at this stage. A NACK for flits of non-selected packets is generated. The third stage is just a multiplexer, which selects the prioritized input port. The following arbitration stage keeps the status of virtual channel registers and decides whether the flits can be stored into the registers or not. A header flit is sent to the register with more free locations, followed by successive flits of the same packet. The fifth stage is the actual buffering stage, and the ACK/NACK message at this stage indicates whether a flit has been successfully stored or not. The following stage takes care of forward flow control and finally a last arbitration stage multiplexes the virtual channels on the physical output link.

$\times pipes$ has been completely described in synthesizable SystemC at the cycle-accurate and signal-accurate level.

4 Network characterization

In a contention-free scenario, latency LT incurred by outstanding flits in $\times pipes$ can be expressed as:

$$LT = SW_LT + LN_LT + NI_SR_LT + NI_DST_LT \quad (1)$$

SW_LT is the switch latency, i.e. switch pipeline depth (7) times the number of crossed switches. LN_LT is the link latency (i.e. number of crossed repetition stages). Finally, NI_SR_LT and NI_DST_LT are the latencies incurred by source and destination network interfaces, respectively. For a 2x2 crossbar, $SW_LT = 7 \times 3 = 21$ (3 switches are crossed in the worst case), $LN_LT = 4$ (assuming each link

consists of 2 stages), $NI_SR_LT = 6$ and $NI_DST_LT = 10$, therefore $LT = 41$ clock cycles are needed to reconstruct the first request at the destination node.

The throughput achievable by our network is one flit per cycle. However, this ideal performance cannot be always exploited because synchronizing the network protocol with the end-to-end OCP protocol (e.g., packetization) gives rise to a bounded but unpredictable number of latency cycles.

The aggregate bandwidth our switch implementation can provide amounts to 64 Gbit/s, for a 32 bit link, 4 output ports and 500 MHz clock frequency. For the same configuration, we finally estimated the buffering resources required by NoC components: about 12 Kbit for the network interface and 3 Kbit for a switch.

5 Conclusions

This paper illustrates $\times pipes$, a NoC targeting both homogeneous and heterogeneous MP-SoC architectures. $\times pipes$ consists of soft NoC macros, so that NoC instance-specific designs can be derived from a general purpose architectural template. Switches provide for reliable and latency insensitive communication. The architecture is inherently scalable and optimized for high throughput and low latency.

References

- [1] D. Culler, J. Pal Singh, A. Gupta, *Parallel Computer Architecture, a hardware/software approach* Morgan Kaufman, 1999.
- [2] K. Compton, S. Hauck, "Reconfigurable Computing: A Survey of Systems and Software", *ACM Computing Surveys*, vol. 34, no. 2, pp. 171-210, June 2002.
- [3] R. Tessier and W. Burleson, "Reconfigurable Computing and Digital Signal Processing: A Survey", in *Journal of VLSI Signal Processing*, vol. 28, no. 3, pp. 7-27, May/June 2001.
- [4] L. Scheffer, "Methodologies and tools for pipelined on-chip interconnect," *Int. Conf. on Computer Design*, pp. 152-157, 2002.
- [5] V. Agarwal, M.S. Hrishkesh, S.W. Keckler, D. Burger, "Clock rate versus ipc: the end of the road for conventional microarchitectures," *Int. Symp. on Computer Architecture* pp.248-250, June 2000.
- [6] L.P. Carloni, K.L. McMillan, A.L. Sangiovanni-Vincentelli, "Theory of latency-insensitive design," *IEEE Trans. on CAD of ICs and Systems* pp.1059-1076, Vol.20, no.9, Sept.2001.
- [7] L. Benini and G. De Micheli, "Networks on chips: a new soc paradigm". *IEEE Computer*, 35:70-78, January 2002.
- [8] D. Wingard, "MicroNetwork-based integration for SOCs," *DAC 2001* pp.673-677, June 2001.
- [9] W.J. Dally, S. Lacy, "VLSI architecture: past, present and future," *Conf. Adv. Research in VLSI* pp.232-241, 1999.
- [10] S. Kumar et al., "A network on chip architecture and design methodology," *ISVLSI 2002*, pp.105-112, 2002.
- [11] J. Walrand, P. Varaja, *High Performance Communication Networks*. Morgan Kaufmann, San Francisco 2000.
- [12] S. Ishiwata et al., "A single-chip MPEG-2 codec based on customizable media embedded processor," *IEEE JSSC*, pp.530-540, Vol.38, no.3, March 2003.
- [13] E. Rijpkema et al., "Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip" *DATE 2003*, pp.350-355, 2003.
- [14] H. Zhang et al., "A 1V Heterogeneous Reconfigurable DSP IC for Wireless Baseband Digital Signal Processing," *IEEE Journal of SSC*, pp.1697-1704, Vol.35, no.11, Nov.2000.
- [15] <http://www.ocpip.org/home> OCP specification