Asynchronous on-chip networks

M. Amde, T. Felicijan, A. Efthymiou, D. Edwards and L. Lavagno

Abstract: Various kinds of asynchronous interconnect and synchronisation mechanisms are being proposed for designing low power, low emission and high-speed SOCs. They facilitate modular design and possess greater resilience to fabrication time inter-chip and run-time intra-chip process variability. They can provide a solution for low power consumption in chips and simplify global timing assumptions, e.g. on clock skew, by having asynchronous communication between modules. A few methodologies, including globally asynchronous, locally synchronous and desynchronisation, aim at leveraging the benefits of both synchronous and asynchronous design paradigms. The authors survey various methodologies used for leveraging asynchronous on-chip communication. They investigate various GALS based implementations, desynchronisation strategies and asynchronous network-on-chip (NoC) designs.

1 Introduction

The main idea of a SoC design methodology is to 'divide' complex chips into several independent functional blocks and 'conquer' each of them using standard synchronous methodologies and existing CAD tools These functional blocks are then connected by means of an on-chip communication infrastructure to form a functional system.

Dividing a chip into smaller blocks keeps the technology scaling problems, such as clock-skew, manageable; however, this is only true for each individual block, while the problems aggravate drastically for the interconnect itself. This is because the network elements may be scattered all over the chip connected by relatively long wires, which do not scale well in deep sub-micron technologies [1]. Synchronising such a network with a single clock source is problematic at best.

There are major problems in having various synchronous on-chip communications, namely:

- *Modularity and design reuse:* In the synchronous world, a complete redesign on the chip is needed if a component of the chip is modified or if the frequency of operation is changed, thus making the design nonmodular. Normally, all the components have to be redesigned at the same new clock frequency. This leads to waste of design effort. GALS IP cores with asynchronous interfaces would make them amenable for design reuse.
- *Electromagnetic interference (EMI):* All the switching activity in a synchronous chip takes place at a given clock tick, making the circuit prone to EMI effects. In comparison, switching activity is distributed over time in an clockless chip.
- *Worst case performance:* The circuit always designed for the worst-case performance, since the critical path in the circuit determines the clock period.
- *Clock power consumption:* Large clock buffer trees present in current design lead to a high power consumption. Studies show that high-speed processors have power consumption dominated by clock and the average clock power consumption by the clock is 45% of the total power consumed [2]. Similar statistics are reported for high- and medium-speed ASICs as well.
- *Clock skew:* The problem of distributing the global clock in a chip with minimal clock skew is getting difficult to solve due to increase in clock frequencies, smaller feature sizes and growing design complexities. Few ASIC designers can afford the sophisticated calibration techniques used in leading edge microprocessors [3], and would like to enjoy the intrinsic robustness with respect to manufacturing and run-time variability that asynchronous circuits exhibit.

Owing to the above-mentioned problems in using a synchronous design style, efforts are being made to design chips asynchronously. A significant advantage of asynchronous design is smoother handling of both fabrication-time inter-chip and run-time intra-chip variability (the later requires completion detection, the former only delay matching). Also, all the aforementioned problems associated with distribution of global clock over the entire chips, clock power consumption, clock skew and EMI, are eliminated. Moreover, the designs become modular since timing assumptions are explicit in the handshaking protocols. Hence no redesign is needed if an asynchronous component is modified. Furthermore, the circuit would work faster, exploiting average case rather than worst case performance.

However, asynchronous design strategies also come with their own set of problems. Asynchronous design is a more difficult task compared to synchronous design. Glitch-free circuits have to be generated as compared to the synchronous domain, where the data only have to be stable before the arrival of the clock. Also, asynchronous design suffers from absence of industrial tool support. Lack of a mature tool flow has prevented this methodology from being widely adopted by designers in industry.

© IEE, 2005

IEE Proceedings online no. 20045093

doi: 10.1049/ip-cdt:20045093

Paper first received 19th July and in revised form 4th October 2004

M. Amde is with the Department of Electrical & Computer Engineering, University of California at San Diego, USA

T. Felicijan, A. Efthymiou and D. Edwards are with the Department of Computer Science, University of Manchester, Oxford Rd, Manchester M13 9PL, UK

L. Lavagno is with Politecnico di Torino, Dipartimento di Elettronica, Corso Ducadegli Abruzzi 24, Torino 10129, Italy

E-mail: luciano@cadence.com

Moreover, several asynchronous circuit implementation techniques have a very high overhead in terms of area, delay and possibly even power consumption. This is due to the fact that truly asynchronous datapaths require implementing each signal in dual rail, and collecting acknowledgments from every gate output in the circuit. In this paper we survey techniques that avoid such large overhead, at the expense of fewer gains in terms of, for example, EMI and average case performance.

The globally asynchronous locally synchronous (GALS) and desynchronisation design styles that are described below are aimed at filling the gap between the purely synchronous and asynchronous domains. GALS consists of synchronous modules on a chip communication asynchronously as shown in the system level view in Fig. 1. These methodologies are promising because they allow synchronous design of components at their own optimum clock frequency, but facilitate asynchronous communication between modules. This leads to a design flow fairly similar to the synchronous flow but with a few additional components which enable asynchronous communication. It eliminates the global clock leading to a huge reduction of power consumption and alleviating the clock skew problem. It facilitates modular system design which is scalable. Close resemblance to synchronous design also makes it amenable to attract the attention of synchronous designers who are not willing to experiment with asynchronous design.

GALS refers to a communication framework in which local clocks are either unsynchronised or paused. This means that there is a risk of meta-stability at the interfaces which is not present in 'traditional' speed-independent or delay-insensitive asynchronous circuits. Metastability is a condition where the voltage level of a signal is at an intermediate level — neither 0 or 1 — and which may persist for an indeterminate amount of time.

Desynchronisation bears some similarity to GALS techniques, in that the datapath remains essentially synchronous and its clocks are locally generated, but it prevents metastability completely by using handshakes. As such, a desynchronised circuit can be obtained automatically from a synchronous one. It has approximately the same area, power and performance, but has lower EMI, due to the spreading over time of clock edges, and better modularity, due to the explicit handshakes between components that automatically satisfy local timing constraints.

In this paper we first present formal frameworks for the analysis of transformations from synchronous to asynchronous systems, and their implementation in the desynchronisation flow. We then proceed to explain various schemes for implementing GALS based systems. We finally conclude with a discussion of asynchronous NoCs, and with a case study.

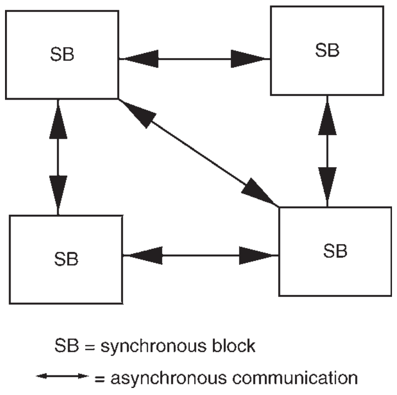

Fig. 1 *System level view of GALS*

2 Formal models

2.1 Multi-clock Esterel

Synchronous design tools have a wide range of tools, giving rise to a tried and tested design flow. Asynchronous circuits suffer from lack of mature design flow, and efforts are being made to capture the asynchronous behaviour of the GALS system in the synchronous domain. One effort in this direction is of multi-clock Esterel [4].

Synchronous languages [5, 6] have a significant advantage with their ability to prove correctness of the hardware circuit before they are actually implemented. Esterel is a synchronous language used for modelling reactive systems interacting with the environment. It is an imperative language and hence uses variables which retain their value until updated. It is used mainly for modelling controller applications and provides synchronous parallelism. Hence it could also be used for modelling hardware systems. Esterel inherently assumes a global clock and it cannot handle a system with multiple clocks.

Multi-clock Esterel provides a framework for modelling multiple local clocks as well as enabling asynchronous communication between various components in the design. It also provides a clean model for integrating Verilog/VHDL features in a design. It aims to retain the existing features of reactive languages like pre-emption and more importantly verifiability. It can be considered to satisfy the 'synchrony hypothesis' as its reactions can be associated with local clock ticks.

The asynchronous communication between concurrently running locally clocked reactive components is based on latches with limited memory. In [4] the authors show an example design of a Micropipeline in a modular fashion and show that multi-clock Esterel modules could be composed in a hierarchical fashion.

Multi-clock Esterel could also be used to model a sub-sets of VHDL code enabling the possibility of hardware–software codesign using VHDL while verifying the entire design in the synchronous paradigm of multi-clock Esterel.

2.2 Signal/polychrony framework

The goal of this research is to model GALS in a multi-clock synchronous environment and map it to an asynchronous system, preserving all the properties proven in the synchronous domain.

Signal [5] is a programming framework which provides a formal way of modelling various synchronous components running on different clocks and validating that the asynchronous composition of the various components would lead to a functionally correct behaviour. It achieves this by transforming the asynchronous composition of the various synchronous components having different clocks to a fully synchronous multi-clock model preserving behavioural equivalence. The synchronous model takes advantage of verification tools available for the synchronous languages. The correct behaviour can thus be checked by extensive simulation and model-checking in the synchronous domain.

This methodology could be used in integrating various IP cores designed at different clock frequencies using a desynchronisation protocol and formally verifying the functional correctness of this GALS network.

In [7], the authors provided a formal way of capturing asynchrony in the synchronous framework of Signal.

They prove that an ideal asynchronous model can be completely mapped in Signal with unbounded FIFOs for inter-component communication. They also show that the class of synchronous models that can be implemented asynchronously without any loss in semantics, i.e. while preserving the deterministic behaviour that is a key characteristics of synchronous models, must satisfy the properties of endochrony and isochrony. Roughly speaking, endochrony means that a component whose interface is going to be made asynchronous must be able to tell from the values of its inputs which inputs must be read next. This approximately corresponds to the sufficient property stated by Kahn [8] to ensure determinate behaviour for the data flow networks, namely that processes cannot probe input FIFO for the presence of data. Isochrony, on the other hand, means that, if two components share a variable, they must agree on the values which are assigned to it at each step. Note that, unfortunately, the identification of bounds to the size of FIFO channels in Kahn Process Networks is undecidable [9], and hence the problem of correctly deploying an arbitrary synchronous system onto an asynchronous architecture must be solved by a human, using a lot of simulation, iteration and guesswork.

The high-level system specification is transformed into a low-level circuit representation through a series of steps. At each step, the transformation from a higher to lower level of abstraction should preserve the correctness across the transformation. Polychrony [10] is a platform which, along with the synchronous programming framework of Signal, provides formal refinement of multi-clocked models from high-level behavioural specification to the low-level synthesis and implementation of these models using formal verification techniques. Polychrony takes a high-level SystemC/SpecC specification and refines it in a semantic-preserving manner towards a GALS implementation. This allows one to leverage the implementation of various synchronous components with multiple clocks with assurance of a functionally correct asynchronous communication between different clocked synchronous components.

The advantage of using Polychrony in a high-level design flow is that it automates the complex task of formal design verification at each stage of refinement and renders the low-level implementation formally correct. The polychronous model of Signal formally captures the behavioural abstractions from SystemC/SpecC programs as well as behavioural specifications from IP cores. The Polychrony platform aids in automating the refinement of behavioural specification towards synthesis while formally verifying the correctness of the transformation at each design flow step. Hence one can rapidly codesign hardware/software GALS architectures while being assured of formally conforming to the original behavioural specifications.

2.3 Desynchronisation

Desynchronisation [11] builds on these theoretical foundations in order to provide the designer with the option to derive a medium-grained asynchronous implementation from a traditional synchronous specification. Assuming an initial design implemented with edge-triggered flip-flops, it requires the following steps:

(i) conversion of the flip-flop-based synchronous circuit into a latch-based one (*M* and *S* latches in Fig. 2*b*)
(ii) generation of matched delays for combinational logic rounded rectangles in Fig. 2*b*
(iii) interconnection of controllers for local clocks.

The method for desynchronising an arbitrary netlist relies on composition of the controllers. It requires to identify direct connections, via combinational logic, between adjacent groups of latches, and then the overall clock generation circuit is obtained through composition of timing diagrams corresponding to these partial descriptions.

The specification of a pairwise interaction between even–odd and odd–even latches for overlapping desynchronisation is shown in Fig. 3. It models the communication of data from latch A to latch B. The latches are transparent when the control signal is high. Initially, only half of the latches contain data ($\mathcal{D}$). Data items flow in such a way that a latch never captures a new item before its successor latches have captured the previous one.

Data overwriting can never occur, even though the pulses for the latch control can overlap. This model is based on

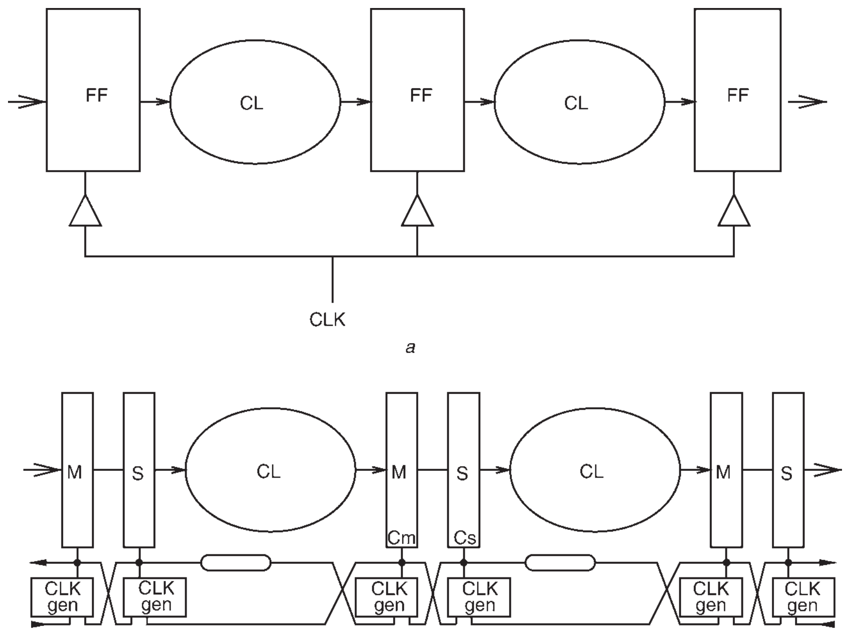

Fig. 2 *Synchronous and desynchronised pipelined circuit*

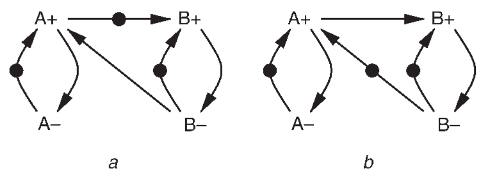

Fig. 3 *Synchronisation between latches*

a Even → odd
b Odd → even

Table 1: Synchronous against desynchronised DLX

	Sync. DLX	Desync. DLX
Cycle time, ns	4.4	4.45
Dyn. power cons., mW	70.9	71.2
Area, μm^2	372 656	378 058

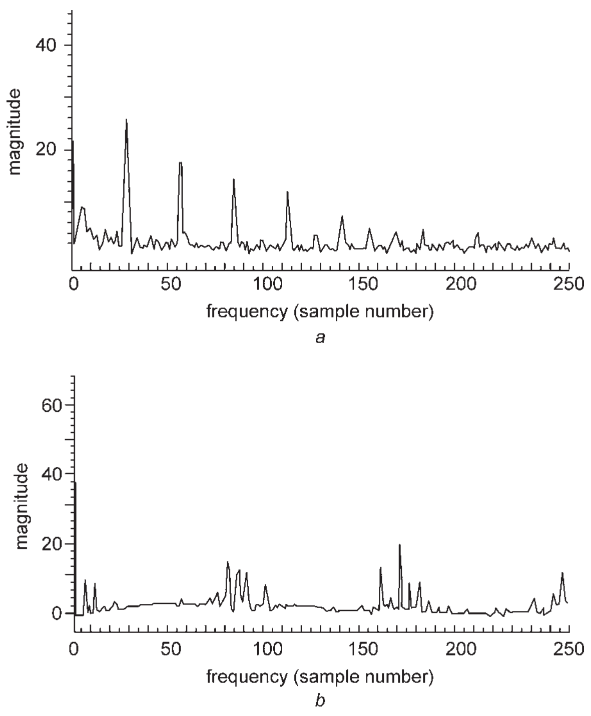

Fig. 4 *FFT of current consumption in synchronous and desynchronised DLX*

a Synchronous
b Desynchronised

the observation that a data item can ripple through more than one latch, as long as the previous values stored in those rippling latches have already been captured by the successor latches. As an example, event $B+$ can fire as soon as data are available in A (arc $A+ \rightarrow B+$) and the previous data in B have been captured by C (arc $C- \rightarrow B+$).

Reference [11] suggests that desynchronisation results in circuits with almost identical area, performance and power consumption as the original synchronous ones. Desynchronised circuits, however, have smaller EMI due to the out-of-phase clocks, and better modularity due to the explicit handshakes encapsulating timing constraints. A comparison between a synchronous and a desynchronised version of the same processor is shown in Table 1.

The electro-magnetic emission advantages can be seen by looking at the spectrum of the current absorbed by the circuit from the power rails, shown in Fig. 4.

3 Mixed synchronous/asynchronous solutions

The Pentium 4™ processor [3] uses 47 different clock domains, whose skew relative to a global reference clock is programmable. Domain clocks were intentionally skewed to improve operating frequency, and up to one speed bin improvement is reported. The design uses two PLLs – one for the core and one for the I/O logic. From these, six different clock frequencies are derived. The Pentium 4™ also has critical portions (e.g. the ALU) working at twice the clock frequency of the rest of the chip [3]. Noncritical ones work at half the clock frequency, in order to save area, power und design effort.

The Alpha processor [12] illustrates the need for flexible clocking schemes in order to enable core reuse in system-on-chip (SOC) designs. The entire chip is partitioned into 11 clock domains, where one domain is a migration of a processor core from an older design. The existing clock distribution in this embedded core is used as a reference clock. Four major clocks (one reference and three derived) are used to clock separate chip sections. Delay-locked loops (DLLs) are used to maintain small phase alignment errors among major clocks.

An example of mixed synchronous and asynchronous implementation is given in [13], which presents the design of a digital FIR filter used in read channels of modern disk drives. The degree of pipelining in the filter is dynamically variable and depends on the input data rate. The performance of this filter was found to be better than existing read channel filters.

The high-speed asynchronous portion of the chip is sandwiched between two synchronous portions. The asynchronous datapath in the chip uses dual-rail dynamic logic and the synchronous datapath in the chip uses single-rail static logic. The asynchronous section relies upon handshakes for communication, whereas the synchronous section is dependent on global clocking. Thus, the interface circuitry between asynchronous and synchronous datapaths is responsible for data conversion. It also needs to adapt to different control signals on either side of the interface. The first interface requires conversion from the synchronous to the asynchronous domain and the second interface requires asynchronous to synchronous conversion. The interface circuitry achieves this by having special latches for performing data conversion and pulse generators for implementing the handshaking protocol for the asynchronous section. In order to resynchronise and avoid metastability at the second interface, a delayed version of the Req handshake signal generated at the first interface is passed directly to the second interface using a programmable delay element. The programmable delay should be greater than the delay for the correct data computation by the asynchronous section.

4 Pausable clock interfacing schemes

Pausable clocking schemes are proposed as mechanisms for data transmission between synchronous modules running at different clock frequencies. In this scheme, the receiver clock is paused whenever the sampling of data lines by the receiver could lead to potential metastability. The sender clock is paused till the data is correctly sampled by the receiving module. This avoids synchronisation failure at the receiving end and flow control at the sender end.

A similar approach is also followed by recent work on the Razor processor [14], in which a comparator (including a meta-stability detector) identifies when a register incorrectly latches a value, due to a critical timing problem.

In the next clock cycle the pipeline is simply restarted with the correct data copied back in every register from a shadow latch, and processing continues synchronously, with 'skipped' clock cycles. Pausable clock schemes listed below, on the other hand, generally stretch clock cycles and do not ensure phase alignment with an external reference clock. Razor is a very promising approach to tackling variability, in that it allows one to clock a processor very close to its true speed. However, reliable operation over extended periods of time, despite the inherent risk of meta-stability, still needs to be demonstrated.

In [15], the authors comment that previously proposed schemes [16] do not scale well for high clock frequencies of locally synchronous (LS) components and multiple cycle delay in clock distribution due to large clock buffer trees. Due to the presence of large clock buffer trees in the LS components, the assumption of previous schemes of data transfer being stalled within one clock cycle of pausing the sender clock does not hold and leads to extra transmissions in what the authors call the 'clock overrun window', which denotes the skew between pausing the clock and actual stopping of data transmission by the sender module.

They propose a circuit for interfacing two high frequency LS modules using a partial handshake protocol which achieves high data rates and has a small probability of failure. A partial handshake is used as it provides faster data transfer than a complete handshake protocol. They propose a 'direct path' FIFO to account for long interconnect delay and an additional 'buffered' FIFO to capture data transferred in the clock overrun window. This scheme does not pause the receiver clock for synchronisation but pauses the sender clock to achieve flow control.

In [15], transistor level sender and receiver interface circuits are given and the models are verified by SPICE simulation. The timing analysis of the interface circuits proves that under certain circumstances of bad signal timings of the signal with respect to sender and receiver clock, the synchronisation circuit would fail with a small probability of failure, thus improving on previous schemes.

Chakraborty *et al.* in [17] discuss using abstract timing diagrams to reason about the correctness of interfacing techniques between synchronous modules. They point out that there are various different interfacing techniques available but it is difficult to compare them due to differences of analysis carried out for each of them. Abstract timing diagrams are used for analysing specific interfacing schemes and understands why certain schemes work under restricted conditions and fail otherwise. The authors further point out that robust asynchronous interfaces could be built if certain new circuits could be implemented.

5 GALS implementations

According to the GALS methodology used in [18], the asynchronous circuits required to convert LS modules to conform to the GALS standard are restricted to implementing 'self-timed' wrappers around each module. Each LS module is driven by a pausable clock in its self-timed wrapper, avoiding metastability and data corruption. The self-timed wrappers consist of a pausable local clock generator, port controllers and test structures as shown in Fig. 5. They have implemented five wrapper elements in technology-independent VHDL. The port controllers are implemented as asynchronous finite state machines using the extended burst mode paradigm of [19]. These are synthesised using the 3-D tool, which results in a synthesisable And–Or implementation [20].

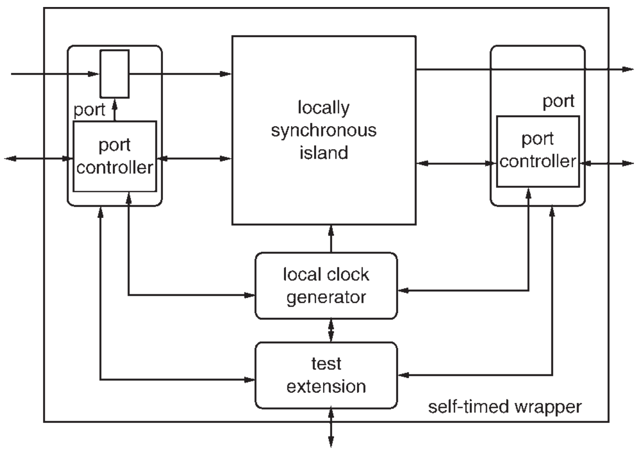

Fig. 5 *Self-timed wrapper*

In [18] the authors describe two different types of port controllers:

(*a*) *Poll-type, or nonblocking, port:* This port is used whenever a data item is needed but computation could proceed without it arriving immediately. The LS modules keeps functioning while the data transfer is handled by the port.
(*b*) *Demand-type, or blocking, port:* This port is used when the LS module cannot continue computation till the arrival of data on the port. While waiting for data, the demand-type port suspends the local clock, reducing power consumption of the module.

Various tunable local-clock generators are compared in [21]. The current research is directed towards highly frequency tunable local oscillators for better performance of individual GALS modules.

Point-to-point interconnects have been implemented to allow asynchronous communication between GALS modules [22]. In [22], interface wrapper circuits are presented for communication between LS modules. The wrapper interface consists of an arbiter and a calibrated delay line. This ensures that a stable local clock signal is generated. Metastability is avoided as clocking is done only after the data are ready. Circuits with sleep mode, where the local clock is stopped due to unavailability of data, are also presented. This could lead to reduced power consumption. FIFOs with various depths have been used as asynchronous channels between modules. The simulations show that designing for FIFO depths greater than 2 does not improve bandwidth of communication between modules.

Multi-point interconnects are also required for efficient SoC GALS systems. Two different interconnection topologies have been proposed in [23]:

(*a*) *GALS bus:* In this architecture all port controllers of LS modules (master/slave) in a design are connected to the same bus, as shown in Fig. 6. The arbitration and address decoding is central for power efficiency.
(*b*) *Ring structure:* A transceiver is associated with the port controller of every LS module, as shown in Fig. 7. The arbitration is done by the transceivers and it decides which request to grant, either from the previous transceiver or from its own port controller, when it wants to insert some packets into the ring.

The ring structure leads to higher latency as each packet has to encounter one or more transceivers, but it leads to lower interconnect length between modules and possibly reduces power consumption. One aspect that is not discussed by

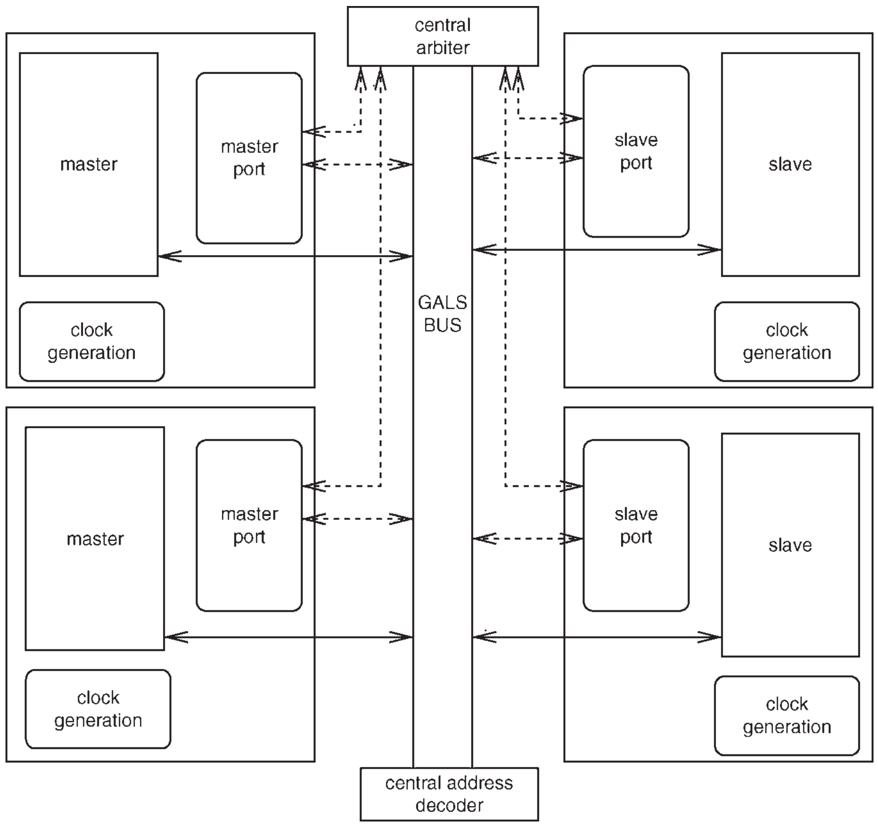

Fig. 6 *GALS bus architecture*

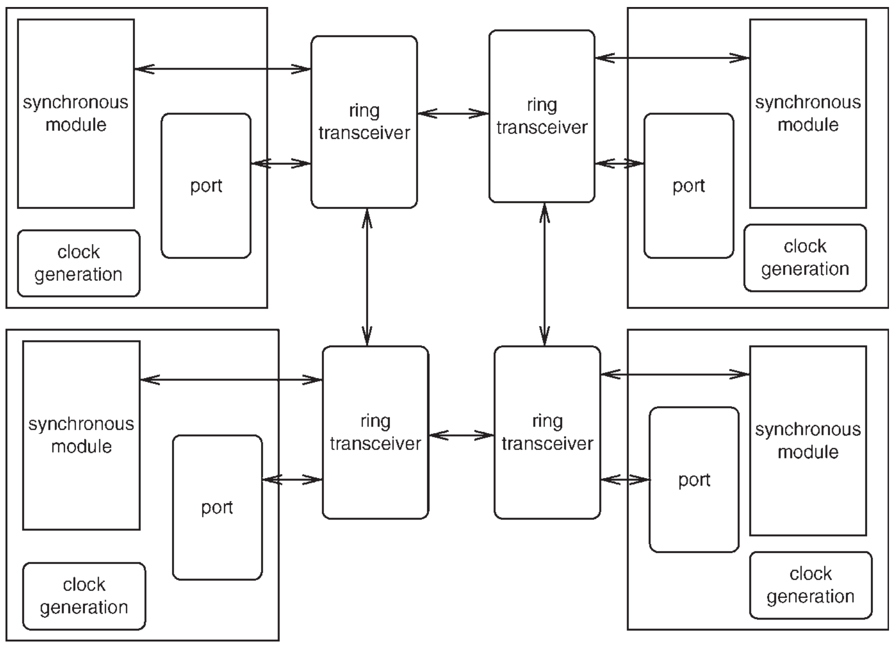

Fig. 7 *Ring architecture*

the authors of [23] is fault tolerance. Failure of one module in a ring architecture would lead to failure of communication between all modules, whereas the GALS bus would be more tolerant to such faults.

A GALS test chip with 3 million transistors was implemented in 0.25 micron technology [24]. The chip contains 25 GALS modules and occupies a total area of 25 square mm. A design flow has been presented for automating the design of GALS chip. This is facilitated by using a library of self-timed elements which can be used to convert synchronous modules to GALS modules. The requirement of a pausable local clock led to the addition of a programmable delay element and Mutex element to the standard cell library. Timing verification was carried out hierarchically at three levels:

(i) inside the self-timed wrapper library
(ii) within each GALS module
(iii) for the handshake signals between GALS modules.

The authors claim that since their GALS methodology requires a limited number of self-timed sub-circuits, most of the design process can be handled by using customised design automation scripts.

In [25], a study has been carried out to measure the performance and power consumption of GALS methodology for a hypothetical super-scalar processor architecture. The results show that GALS design does not lower power consumption appreciably and the overheads of using multi-clocked synchronous blocks leads to a performance drop in the range of 5–15%. It further says that voltage scaling techniques for each synchronous block would help bridge the performance gap. The authors of [26] propose a strategy for optimally partitioning the synchronous logic into synchronous blocks for maximising power reduction. They report average power reduction of 30%.

6 Current research in asynchronous NoC

Employing totally self-timed techniques for the interconnect is, as mentioned in Section 1, a promising means to tackle a number of on-chip interconnection issues, from power and EMI reduction to clock skew management, to modularity of design. However, only a few proposals for an asynchronous NoC have been published so far. This Section gives an overview of the state-of-the-art research in asynchronous on-chip networks.

CHAIN (chip area on-chip interconnect) was designed by Bainbridge at the University of Manchester, UK [27]. The network is based on narrow delay-insensitive high-speed links using one-of-five data encoding combined with a return-to-zero signalling protocol. In a 0.35 micron VLSI technology the author claims that a single chain link provides a throughput of around 700 Mbps and more than 1 Gbps in 0.18 micron CMOS technology using suitable link lengths to minimise end-to-end latency. To increase the bandwidth, multiple links can be bundled together to form a wider data path.

CHAIN does not require a fixed network topology but allows a designer to adapt the topology of the network to a specific SoC using three basic network elements: a router, an arbiter and a multiplexer. To further improve the flexibility of the network, source routing is employed with a variable length packet organisation. The routing information is encoded in a series of routing symbols at the start of every packet. The length of a packet is designated by the EOP (end-of-a-packet) symbol, which also has a function to tear down the route set by the header of the packet.

CHAIN implements a split transaction protocol typically employing two separate networks for the command and response in order to improve the performance of the interconnect. Also, the network supports atomic sequences of multiple commands.

NEXUS is another asynchronous on-chip network developed at Fulcrum Microsystems, USA [28]. Their approach is based on a 16-port, 36-bit asynchronous crossbar that connects synchronous modules through asynchronous channels and clock-domain converters. Nexus is a quasi-delay-insensitive (QDI) on-chip interconnect infrastructure using one-of-four encoding and pre-charge domino logic. It also supports a split transaction protocol with a request burst going out and a completion burst returning. Implemented in a 0.13 micron low-voltage CMOS process, Nexus runs at 1.35 GHz and exhibits the latency of 2 ns.

Liljeberg *et al.*, from the University of Turku, Finland, propose a self-timed ring architecture as a replacement for on-chip buses [29]. They implemented a 12-stage bi-directional ring network with 36 pipeline sections. The network employs a two-phase signalling protocol between stages to accommodate relatively long wire segments with less transitions within a transaction cycle, and a four-phase signalling protocol for internal control within a stage to enable design of fast and relatively simple control logic circuitry. The data path is encoded using a standard single-encoding scheme.

The authors compared three closely related structures: a bi-directional ring, a bi-directional folded ring and a bi-directional open ring against different types of traffic. The simulation results show that the peak throughput of a single segment in one direction is between 0.8 and 1.0 Gwords/s in 0.18 micron technology with the segment lengths of 1 and 4 mm, respectively. The maximum measured throughput of the whole ring is 6.61 Gwords/s.

An asynchronous ring-based network was also proposed in [30].

A related research direction is that of asynchronous communication mechanisms (ACMs [31]), which implement various degrees of synchronisation between communicating parties, from fully independent to fully interlocked. ACMs are an important mechanism, especially in the form which does not block the reader nor the writer of a communication channel, in order to implement hard real-time with asynchronous techniques. In other words, they go beyond QoS-based soft real-time, in order to provide full timing guarantees to safety-critical systems. The cost is slightly higher than traditional FIFO-based mechanisms, which block the reader when empty and the writer when full, and the performance is comparable to that of traditional FIFOs [32].

7 QoS for NoCs

A modern SoC may consist of many different components and IP blocks interconnected by an NoC. These components can exhibit disparate traffic characteristics and constraints, such as requirements for guaranteed throughput and bounded communication latency.

As an example, consider a connection between a video camera and an MPEG encoder. Such a connection has to maintain a constant throughput with bounded jitter (variation in end-to-end latency) in order to support the required quality of the system. If the camera and the encoder are a part of a complex SoC interconnected by an on-chip network, the connection has to share the network bandwidth with the rest of the traffic. In order to maintain the quality of the system, the network has to provide the required bandwidth for the connection at any given time.

It is therefore essential for a modem NoC to support quality-of-service (QoS) in order to accommodate such components sharing the same communication medium. Furthermore, the ability of an NoC to provide guaranteed services enables a designer to make critical timing decisions early in the design process, thus avoiding unnecessary design iterations [33].

In synchronous networks QoS is often provided by time division multiplexing (TDM). TDM partitions the time axis into time-slots where each time-slot presents a unit of time in which a single flow can transmit data over a physical channel. Guaranteed throughput is provided by reserving a proportion of time-slots for a particular flow. For example, if a connection requires 50% of the available bandwidth, a network has to ensure that every other time-slot is available for that particular connection. Reserved slots traverse the network in a well synchronised manner without having to arbitrate for the output link with the rest of the traffic. The Aethereal NoC developed at Philips and the Sonics on-chip bus employ a TDM technique to support guaranteed throughput [34, 35].

Although TDM provides a high level of QoS it is unsuitable for asynchronous implementation because it requires global synchronisation between network elements. Another way to provide QoS is to employ a packet scheduling algorithm that will prioritise input traffic in terms of the level of QoS required. In [36], Felicijan *et al.* proposed a QoS architecture suitable for asynchronous on-chip networks using virtual channels [37], where a connection with QoS requirements uses a virtual channel in order to reserve buffer space. The bandwidth of the network is distributed using a priority based asynchronous arbiter according to the priority level of each individual virtual channel. The same authors also proposed a low latency asynchronous arbiter suitable for QoS applications [38] which overcomes the problem of allowing a contender to obtain over 50% of the resource allocation in a self-timed system by using downstream knowledge to trigger the arbitration.

8 Case study: the ASPIDA network-on-chip

ASPIDA (asynchronous open-source processor IP of the DLX architecture) is a project which aims to demonstrate the feasibility of designing and delivering an asynchronous IP in a portable, reusable manner.

With regard to asynchronous networks on chip, one of the main contributions of this project is the creation of an asynchronous interface specification aiming to become the asynchronous equivalent of WISHBONE [39], a synchronous SoC interconnection architecture for reusable IP cores. This interface specification is heavily influenced from both CHAIN and WISHBONE. Most of the interface signals are named following the WISHBONE convention. The major difference from WISHBONE is that the model for inter-core communication is based on split transactions. Thus there are two separate interconnect fabrics: one for commands and another for responses. The asynchronous request–acknowledge handshake signals make this interface specification robust and easy to reuse without a need to verify complex timing assumptions.

ASPIDA will produce a demonstrator chip containing an asynchronous system on a chip. Figure 8 shows its main components, which include an asynchronous open-source DLX processor core (obtained using the desynchronisation techniques described above), two memories dedicated for instructions (IMEM) and data (DMEM), a test interface controller (TIC) for initialisation/debugging, and three interfaces with the external world: a synchronous WISHBONE interface for connection to synchronous peripherals (WB), a novel asynchronous, general-purpose interface (GP), and a 'bare' CHAIN interface (BC) for adding more CHAIN networks and/or debugging the interconnection.

In the ASPIDA system there are three initiators (masters) and five targets (slaves). The DLX core has a Harvard, architecture, so it has two initiators in the interconnection – one for the instruction port and another for the data port.The remaining initiator is attached to the TIC so that it can access the memories and the external interfaces, as well as being able to test the interconnection.

Two of the five targets are the system's memory: 1K words of SRAM each. Their main purpose in the system is to provide fast, on-chip memory space for the DLX so that it can run simple programs at a high speed without the need to

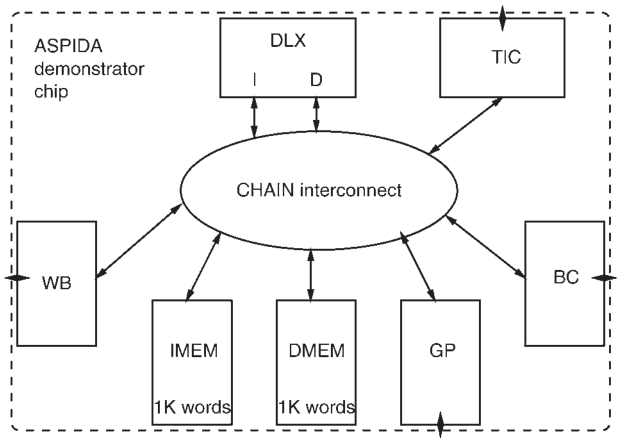

Fig. 8 *Top-level block diagram of ASPIDA demonstrator chip*

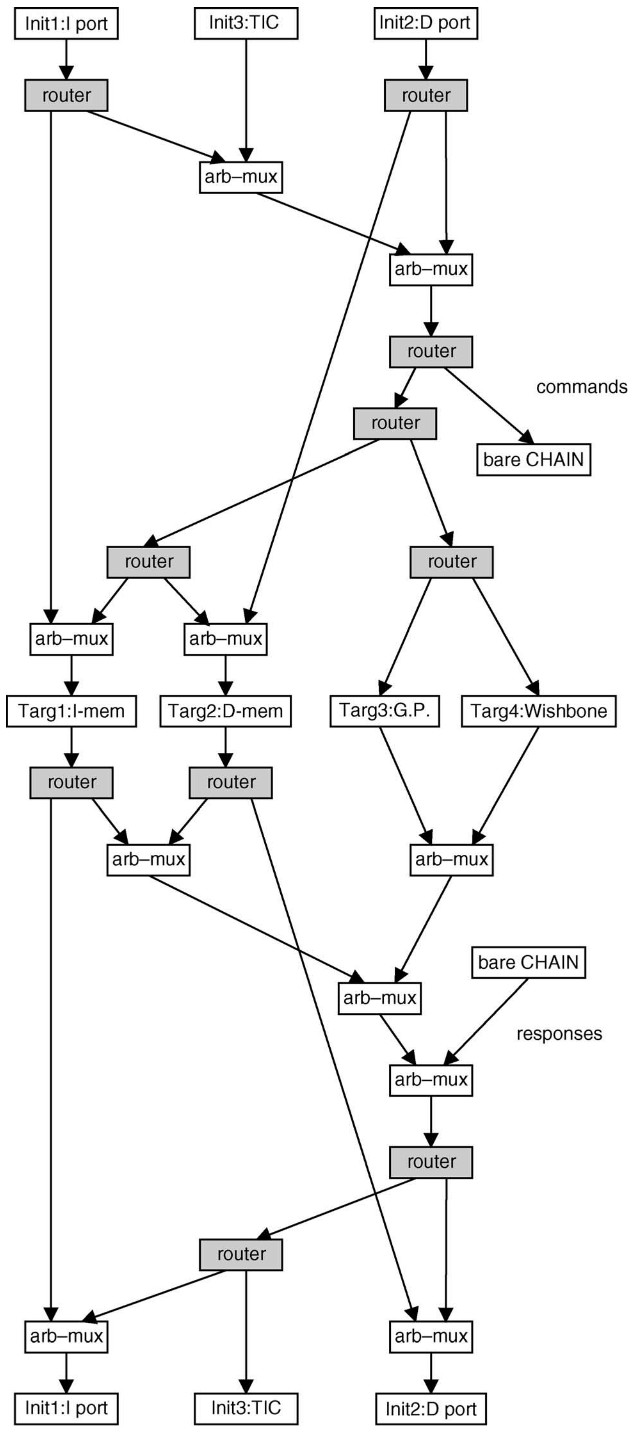

Fig. 9 *ASPIDA interconnection network*

access a slow external memory and without taking up too much area. The remaining targets essentially extend the interconnection off-chip, so that synchronous (WB port) or asynchronous (GP port) 'peripherals' can be accessed by the processor.

In the above system the most common situation is that the processor instruction port will communicate with the instruction memory and the data port with the data memory. All other initiator-target communications should be made possible by the interconnect, but their performance is not as crucial. The above observation led to the interconnection architecture shown in Fig. 9, which optimises the two commonly used paths.

8.1 Design for testability features of interconnect

One of the major challenges of the project was to include circuits that guarantee full test coverage, as a typical synchronous system would. Although well known techniques can be applied to the circuits implementing the processor data-paths and the SRAMs, for the delay-insensitive circuits of CHAIN, these techniques would dramatically increase the area and reduce the circuit speed.

The approach followed for the interconnect fabric is to insert scan-latches in the acknowledge paths of the CHAIN pipeline stages, so that the common input of the parallel C elements can be controllable. In comparison, the standard approach would require a scan-latch for each C clement. Thus our approach resulted in considerable area savings and performance improvement.

The test patterns are manually generated for each of the four basic building blocks of CHAIN interconnects, and a computer program has been developed which, given the network topology and the patterns for each of the component types, produces a full test sequence that gives over 99.5% stuck-at fault coverage for the interconnect. The test strategy is explained in [40, 41].

8.2 Implementation

Since one of the aims of ASPIDA is to produce portable, reusable asynchronous IP, the implementation is standard-cell based, using a 0.18 μm technology. It should be noted that area and performance improvements can be gained by using even a small number of special asynchronous cells.

The processor core is built using the desynchronisation techniques described earlier. The interconnection is drawn as a schematic diagram and later passed on to a standard synthesis tool to optimise the gate-mapping.

For the results presented here, only the network was placed and routed. The area of the network, including the interfaces, is 0.63 mm^2, with the cell density set at approximately 70%. Actually most of the above area is taken up by the interfaces; the actual interconnection fabric occupies $< 15\%$ of the total core area.

8.3 Evaluation

To evaluate the performance of the ASPIDA interconnect, two sets of simulations were conducted, creating different traffic scenarios in the network.

Each master interface is connected to a traffic generator programmable to inject different types of traffic into the network. The masters generate two types of commands, read and write, in the proportion 70% and 30%, respectively. The length of a write command packet is 10 bytes and the length of a read command packet is 6 bytes. Similarly, each slave interface is connected to a dummy client to emulate the behaviour of a slave client. The length of a response packet depends on the type of the command received from a master and represents 6 bytes in the case of a read command and 2 bytes in the case of a write command. Therefore the total number of bytes transferred in a single command–response cycle is 12.

The performance of the network was assessed by measuring throughput and latency of each individual master. The throughput corresponding to the number of bytes a single master is able to transmit and receive per unit of time, and the latency represents the time between when a master sends a command through the network and when it receives the response back from the client. Note that the dummy clients exhibit zero service time, and thus the latency represents only the time packets spent traversing the network.

As mentioned above, two sets of simulations were conducted in order to evaluate the performance of the network. The first set was designed to mimic the traffic characteristics of the environment the network was designed for. In this case masters IP and DP were set to generate commands for slaves IM and DM, respectively, as fast as possible. Furthermore, master TIC was set to generate commands to IM and DM in order to disrupt the throughput of the masters IP and DP. The TIC generates commands at random intervals, according to the exponential distribution function, with an average packet rate between 0 and 100% of the physical bandwidth. Figure 10 shows how the traffic generated by the TIC affects the throughput of the IP.

When there is no TIC traffic present the network dedicates the whole physical bandwidth to the IP master; however, when the TIC traffic is introduced, the IP throughput decreases almost linearly until it reaches around 50% of the maximum bandwidth. Figure 10 shows that the network guarantees approximately half of the physical bandwidth to the IP master. In terms of latency, Fig. 11 shows a similar situation. Note that the masters do not have

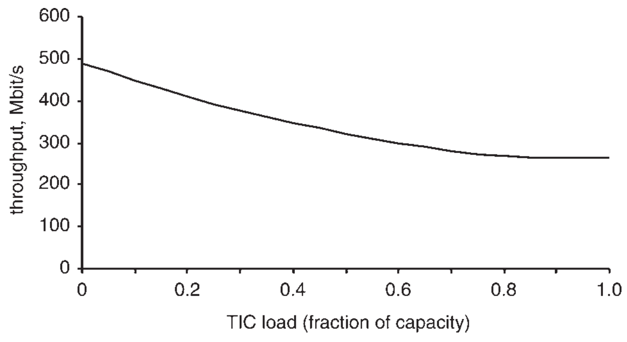

Fig. 10 *Impact of TIC load on throughput of IP*
Note that TIC has same impact on throughput of DP master

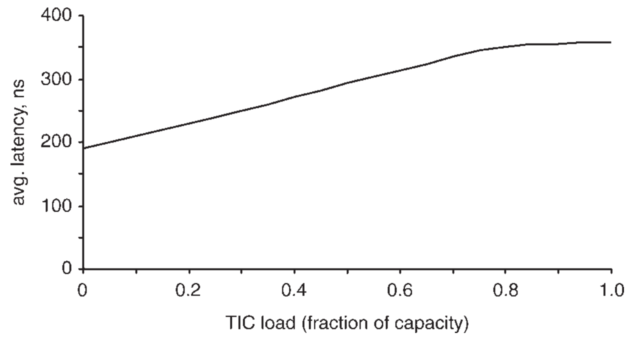

Fig. 11 *Latency of IP master against TIC load*

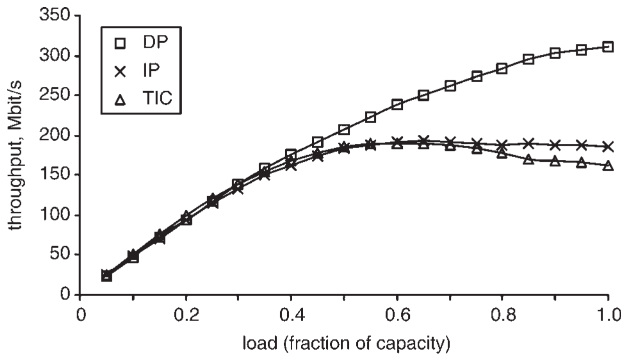

Fig. 12 *Throughput of IP, DP and TIC masters*

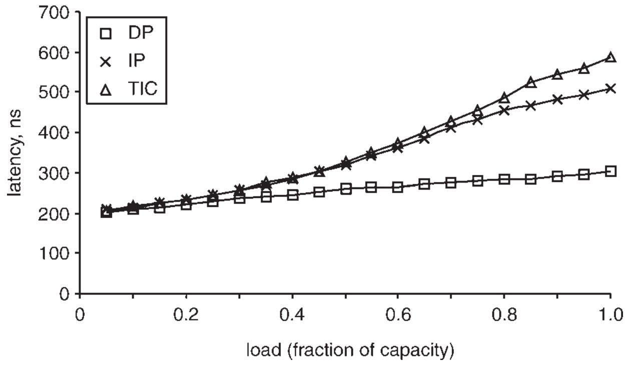

Fig. 13 *Latency of IP, DP and TIC masters*

any buffering capabilities; thus the results shown in Fig. 11 do not include any queueing time.

The second set of simulations generates a more generic traffic scenario in the network. In this case all three masters (IP, DP and TIC) were programmed to issue command packets to randomly chosen slave targets (WB, IM, DM, BC and GP) for every transaction cycle. Furthermore, each master generates commands exponentially distributed across the time axis. The throughput and latency of each master was measured against different traffic loads. Figures 12 and 13 show the throughput and the latency of the IP, DP and TIC masters, respectively.

It is interesting to note that the relative ranking of the three masters in this set of simulations reflects the topology of the network. As the targets are selected randomly, most of the traffic will follow the main trunk of the fabric, as shown in Fig. 9. So IP will be 'fighting' with TIC and the combined flow with DP. Thus the performance of DP is significantly better when the network is congested. Among IP and TIC, the former is connected to IM through a relatively short route, while the latter does not have such special connections. As in one out of five times the IP sends packets to IM, it manifests, on aggregate, a somewhat better performance than TIC.

9 Conclusion

With the SIA roadmap pointing to increasing clock frequencies and smaller feature sizes, distributing a global clock across an entire chip is becoming less and less feasible. While progress in clocking structures continues, several research groups convincingly argue that a complete paradigm shift would ensure significant advantages.

We first surveyed techniques which minimally depart from the synchronous scheme, or in some sense are loosely coupled synchronous schemes, as in the case of desynchronisation. These are the most likely candidates to be picked up first by conservative design teams. They are very easy to use, fully automated or at least tool-supported, but also provide little incentive beyond EMI reduction. The latter is a significant issue only for very cheap integrated circuits, due to the reduced packaging cost, and in security applications, due to the reduced data-correlated emissions.

We then considered stoppable clocks and GALS schemes, which retain a fully synchronous design methodology for the LS blocks, while using standardised wrappers, produced by module generators, for the interfacing. They provide more independence between the modules, in that the overall performance need not be determined by the slowest stage, but may exhibit meta-stability, thus resulting in potentially unpredictable performance.

Finally we looked at truly asynchronous NoCs, which again, due to the need for standardised design flows, use predefined modules and module generators, whose output is then assembled to determine the overall network logic. These asynchronous structures have the best power and performance, but are often less efficient in terms of area, due to the lack of established logic optimisation tools for asynchronous gate-level netlists.

10 Acknowledgments

Aspects of this work were supported by the EU through the IST-2002-37796 ASPIDA project and by the EPSRC under grant GR/R47363/01. The authors would like to thank John Bainbridge for providing many of the circuits used in the case study presented and for useful discussion on the CHAIN interconnection system.

11 References

1 Ho, R., Mai, K., and Horowitz, M.: 'The future of wires', *Proc. IEEE*, 2001, **89**, (4), pp. 490–504
2 Tiwari, V., Singh, D., Rajgopal, S., Mehta, G., Patel, R., and Baez, F.: 'Reducing power in high-performance microprocessors'. Proc. 35th Ann. Conf. on Design Automation, 1998, pp. 732–737
3 Kurd, N., Barkatullah, J., Dizon, R., Fletcher, T., and Madland, P.: 'Multi-ghz clocking scheme for Intel Pentium 4™ microprocessor'. IEEE Int. Solid-State Circuits Conf., 2001, pp. 404–405
4 Rajan, B., and Shyamasundar, R.K.: 'Multiclock ESTEREL: A reactive framework for asynchronous design'. Int. Conf Parallel and Distributed Processing Symp., 2000, pp. 201–210
5 Halbwachs, N.: 'Synchronous programming of reactive systems' (Kluwer Academic Publishers, 1993)
6 Edwards, S., Lavagno, L., Lee, E.A., and Sangiovanni-Vincentelli, A.: 'Design of embedded systems: Formal models, validation, and synthesis', *Proc. IEEE*, 1997, **85**, (3), pp. 366–390
7 Mousavi, M.R., Le Guernic, P., Talpin, J-P., Shukla, S.K., and Basten, T.: 'Modeling and validating globally asynchronous design in synchronous frameworks'. Proc. Conf. on Design Automation and Test in Europe, 2004, pp. 384–389
8 Kahn, G.: 'The semantics of a simple language for parallel programming'. Proc. IFIP Congress, August 1974
9 Buck, J.T.: 'Scheduling dynamic dataflow graphs with bounded memory using the token flow model'. PhD thesis, University of California at Berkeley, 1993, UCB/ERL Memo M93/69
10 Talpin, J-P., Le Guernic, P., Shukla, S.K., Gupta, R., and Doucet, F.: 'Polychrony for formal refinement-checking in a system-level design methodology'. 3rd Int. Conf. on Application of Concurrency to System Design, June 2003, pp. 9–19
11 Blunno, I., Cortadella, J., Kondratyev, A., Lavango, L., Lwin, K., and Sotiriou, C.: 'Handshake protocols for de-synchronization'. Proc. Int. Symp. on Advanced Research in Asynchronous Circuits and Systems. IEEE Computer Society Press, April 2004
12 Xanthopoulos, T., Bailey, D., Gangwar, A., Gowan, M., Jain, A., and Prewitt, B.: 'The design and analysis of the clock distribution network for a 1.2 GHz Alpha microprocessor'. IEEE Int. Solid-State Circuits Conf., 2001, pp. 402–403
13 Singh, M., Tierno, J.A., Rylyakov, A., Rylov, S., and Nowick, S.M.: 'An adaptively-pipelined mixed synchronous-asynchronous digital FIR filter chip operating at 1.3 gigahertz'. Proc. Int. Symp. on

Advanced Research in Asynchronous Circuits and Systems, April 2002, pp. 84–95

14 Austin, T., Blaauw, D., Mudge, T., and Flautner, K.: 'Making typical silicon matter with razor', *Computer*, 2004, **37**, (3), pp. 41–49

15 Mekie, J., Chakraborty, S., and Sharma, D.K.: 'Evaluation of pausible clocking for interfacing high speed IP cores in GALS framework'. Proc. Int. Conf. on VLSI Design, 2004, pp. 559–564

16 Yun, K.Y., and Dooply, A.E.: 'Pausible clocking-based heterogeneous systems', *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, 1999, **7**, (4), pp. 482–488

17 Chakraborty, S., Mekie, J., and Sharma, D.K.: 'Reasoning about synchronization issues in GALS systems: A unified approach'. Proc. Workshop on Formal Methods in GALS Architectures (FMGALS), Formal Methods Europe Symposium, September 2003

18 Gurkaynak, F.K., Villiger, T., and Oetiker, S.: 'An introduction to the GALS methodology at ETH Zurich'. Proc. Formal Methods for Globally Asynchronous Locally Synchronous (GALS) Architecture (FMGALS), September 2003, pp. 32–41

19 Yun, K.Y., and Dill, D.L.: 'Automatic synthesis of extended burst-mode circuits: Part I (specification and hazard-free implementations)', *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 1999, **18**, (2), pp. 101–117

20 Yun, K.Y., and Dill, D.L.: 'Automatic synthesis of 3D asynchronous state machines'. Proc. 1992 IEEE/ACM Int. Conf. on Computer-Aided Design, IEEE Computer Society Press, 1992, pp. 576–580

21 Oetiker, S., Villiger, T., Gurkaynak, F.K., Kaeslin, H., Felber, N., and Fichtner, W.: 'High resolution clock generators for globally-asynchronous locally-synchronous designs'. Handouts of the Second Asynchronous Circuit Design Workshop (ACiD), January 2002

22 Moore, S., Taylor, G., Mullins, R., and Robinson, P.: 'Point to point GALS interconnect'. 8th Int. Symp. on Advanced Research in Asynchronous Circuits and Systems, 2002

23 Villiger, T., Gurkaynak, F.K., Oetiker, S., Kaeslin, H., Felber, N., and Fichtner, W.: 'Mulit-point interconnect for globally-asynchronous locally synchronous systems'. Handouts of the Second Asynchronous Circuit Design Workshop (ACiD), January 2002

24 Oetiker, S., Gurkaynak, F.K., Villiger, T., Kaeslin, H., Felber, N., and Fichtner, W.: 'Design flow for a 3-million transistor GALS test chip'. Handouts of the Third Asynchronous Circuit Design Workshop (ACiD), January 2003

25 Iyer, A., and Marculescu, D.: 'Power and performance evaluation of globally asynchronous locally synchronous processors'. Proc. 29th Ann. Int. Symp. on Computer Architecture, IEEE Computer Society, 2002, pp. 158–168

26 Hemani, A., Meincke, T., Kumar, S., Postula, A., Olsson, T., Nilsson, P., Oberg, J., Ellervee, P., and Lundqvist, D.: 'Lowering power consumption in clock by using globally asynchronous locally synchronous design style'. Proc. 36th ACM/IEEE Conf. on Design Automation, (ACM Press, 1999), pp. 873–878

27 Bainbridge, J., and Furber, S.: 'CHAIN: A delay-insensitive chip area interconnect', *IEEE Micro.*, 2002, **22**, (5), pp. 16–23

28 Lines, A.: 'Asynchronous interconnect for synchronous SoC design', *IEEE Micro*, 2004, **24**, (1), pp. 32–41

29 Liljeberg, P., Plosila, J., and Isoaho, J.: 'Self-timed architecture for SoC applications'. Proc. 16th IEEE Int. SoC Conf., 2003, pp. 359–361

30 Carrion, C., and Yakovlev, A.: 'Design and evaluation of two asynchronous token ring adapters'. Technical report, CS-TR: 562, Department of Computing Science, University of Newcastle, 1997

31 Simpson, H.: 'Four-slot fully asynchronous communication mechanism', *IEE Proc. E, Comput. Digit. Tech.*, 1990, **137**, (1), pp. 17–30

32 Xia, F., Yakovlev, A., Clark, I., and Shang, D.: 'Data communication in systems with heterogeneous timing', *IEEE Micro*, 2002, **22**, (6), pp. 58–69

33 Goossens, K., van Meerbergen, J., Peteers, A., and Wielage, P.: 'Networks on silicon: Combining best effort and guaranteed services'. Design Automation and Test in Europe Conf. (DATE), March 2002, pp. 423–425

34 Rijpkema, E., Goossens, K., Dielissen, J., Rădulescu, A., van Meerbergen, J., Wielage, P., and Waterlander, E.: 'Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip', *IEE Proc., Comput. Digit. Tech.*, 2003, **150**, (5), pp. 294–302

35 Wingard, D.: 'Micronetwork-based integration for socs'. Proc. Design Automation Conf. (DAC), Las Vegas, USA, June 2001

36 Felicijan, T., and Furber, S.: 'Quality of service (QoS) for asynchronous on-chip networks'. Formal Methods for Globally Asynchronous Locally Synchronous Architecture (FMGALS), September 2003

37 Dally, W.J.: 'Virtual-channel flow control', *IEEE Trans. Parallel Distrib. Syst.*, 1992, **3**, (2), pp. 194–205

38 Felicijan, T., Bainbridge, J., and Furber, S.: 'An asynchronous low latency arbiter for quality-of-service (QoS) applications'. Proc. 15th IEEE Int. Conf. on Microelectronics, December 2003, pp. 123–126

39 OpenCores Organization: 'WISHBONE system-on-chip (SoC) interconnection architecture for portable IP cores'. http://www.opencores.org/wishbone/doc/specs/wbspec_b3.pdf, 2002

40 Efthymiou, A., Sotiriou, C., and Edwards, D.: 'Automatic scan insertion and pattern generation for asynchronous circuits'. Design Automation and Test in Europe Conf. (DATE), February 2004, p.672

41 Efthymiou, A., Bainbridge, J., and Edwards, D.: 'Adding testability to an asynchronous interconnect for globally-asynchronous, locally-asynchronous systems-on-chip'. IEEE Asian Test Symp., November 2004, to be published