aelite: A Flit-Synchronous Network on Chip with Composable and Predictable Services

Andreas Hansson[1], Mahesh Subburaman[2], Kees Goossens[3,4]
[1]Electronic Systems Group, Eindhoven University of Technology, Eindhoven, The Netherlands
[2]Electrical Engineering, Linköping Institute of Technology, Linköping, Sweden
[3]Corporate Research Department, NXP Semiconductors, Eindhoven, The Netherlands
[4]Computer Engineering, Delft University of Technology, Delft, The Netherlands
m.a.hansson@tue.nl

***Abstract*—To accommodate the growing number of applications integrated on a single chip, Networks on Chip (NoC) must offer scalability not only on the architectural, but also on the physical and functional level. In addition, real-time applications require Guaranteed Services (GS), with latency and throughput bounds. Traditionally, NoC architectures only deliver scalability on two of the aforementioned three levels, or do not offer GS.**

In this paper we present the composable and predictable aelite NoC architecture, that offers only GS, based on flit-synchronous Time Division Multiplexing (TDM). In contrast to other TDM-based NoCs, scalability on the physical level is achieved by using mesochronous or asynchronous links. Functional scalability is accomplished by completely isolating applications, and by having a router architecture that does not limit the number of service levels or connections. We demonstrate how aelite delivers the requested service to hundreds of simultaneous connections, and does so with 5 times less area compared to a state-of-the-art NoC.

I. Introduction

Systems on Chip (SoC) grow in complexity with an increasing number of *independent applications* on a single chip [1]. The applications are realised by hardware and software Intellectual Property (IP), e.g. processors and application code. The growing number of applications require a system architecture that is scalable on the physical, architectural and functional level [2]. In addition, many applications have requirements on the end-to-end *real-time* behaviour. The system designer has to integrate *all* application IP, from *independent suppliers*, and verify *the combined functional and temporal behaviour*.

Networks on Chip (NoC) have emerged as the design paradigm for scalable on-chip communication architectures. Most NoCs address scalability on the architectural and physical levels [3]–[7], with modular design, Globally Asynchronous Locally Synchronous (GALS) design on the level of independent IPs, and mesochronous or asynchronous communication *within* the NoC. These NoCs, however, offer no or limited scalability on the functional level as applications cannot be developed and analysed in isolation, due to *interference* in shared resources. The interference, although bounded in some cases [3], [4], [6], [7], couples the temporal, and potentially also functional, behaviours of the applications, thus making the burden of verifying application IP the responsibility of the system designer. The complexity of such *monolithic* system analysis grows exponentially and is a major design challenge [2].

NoCs based on Time Division Multiplexing (TDM) [8], [9] completely remove interference between applications, thus providing composable services. They also bound latency and throughput for individual connection, making the services predictable and accommodating applications with real-time requirements. Moreover, in contrast to NoCs based on Virtual Circuits (VC) [3], [4], [6], [7] the speed and area of the NoC is *not* negatively affected by the number of connections sharing a link. TDM-based NoCs are thus scalable on the functional level, but traditionally rely on a notion of *global synchronicity* [8], [9]. This is becoming prohibitively difficult (and costly) to achieve in large chips [3], [10], and is not scalable on the physical level [5].

As the major contribution of this work, we present the *aelite* NoC architecture, which is a light version of the Æthereal [8] NoC, with *mesochronous* or even *asynchronous* links. Like Æthereal, aelite uses TDM-based arbitration, but does so without requiring global synchronicity within the NoC. We demonstrate how the *flit-synchronous* aelite architecture is the first NoC to offer scalability on the physical, architectural and functional level, and how it delivers cost-efficient composable and predictable Guaranteed Services (GS).

The remainder of the paper is structured as follows. We start by introducing related work in Section II. Next, the problem domain is described in Section III. Thereafter, the aelite architecture is described in three steps, starting with the synchronous router in Section IV, followed by the mesochronous link pipeline stage in Section V, and the asynchronous wrapper in Section VI. Finally, experimental results are shown in Section VII and conclusions are drawn in Section VIII.

II. Related work

Many asynchronous [3], [6], [7], mesochronous [4] and globally synchronous [8], [9] NoCs provide some for of GS, by means of latency and throughput bounds for one or more connections. Most common is GS based on VCs [3], [4], [6], [7]. With strict priority arbitration between the VCs [4], [6], [7], only one VC per link can be given bounds on its latency and throughput, due to the lack of rate regulation. Hence, GS connections cannot share links, and in practise, these NoCs only have two service levels, GS and Best Effort (BE).

The fully asynchronous Mango NoC overcomes the problems of strict priority-based arbitration by introducing a rate-regulator [3]. As the router uses VC buffering, it grows with the number of GS connections passing through it, which is not scalable. Additionally, the arbiter can only allocate a fraction of the link capacity. Most importantly, Mango only bounds interference, but does not remove it completely, thus coupling the behaviour of different applications.

Nostrum [9] and Æthereal [8] offer TDM-based GS, and hence composable and predictable services. Both NoCs, however, rely on a globally synchronous NoC. While techniques such as *link pipelining* have been proposed to overcome link latency [11], [12], the cycle-level synchronicity negatively affects the NoC scalability [5].

To overcome the disadvantages of global synchronicity, but still enable a traditional synchronous design style, the NoCs in [4], [5], [13] use mesochronous and asynchronous links [14] between synchronous network elements. However, no GS is offered by [5], [13], and priority-based GS that bounds, but does not remove interference, is provided in [4].

In addition to the aforementioned limitations on physical scalability and provision of GS, none of the works in [3], [4], [6], [7], [9] demonstrate the ability to provide latency and throughput guarantees for more than a handful connections.

We extend existing work by presenting a *flit-synchronous* NoC architecture that provides *TDM-based composable and predictable services*. We show how this architecture enables not only complete isolation of applications and provision of their real-time requirements, but also a significantly smaller and faster design compared to a state-of-the-art NoC.

III. Problem description

We start this section by revisiting the main concepts of the Æthereal NoC [8] that are maintained by aelite, and end with an enumeration of the problems addressed in this paper.

The applications are realised by software and hardware IP, where the latter are connected to the NoC. The IP ports communicate over logical connections, realised by the Network Interfaces (NI), the routers and the physical links that together constitute the NoC. *End-to-end flow control* is used to avoid buffer overflows in the NIs and the path is determined by *source routing*. The NIs interface with the IPs through bi-synchronous FIFOs, thus enabling a GALS design approach. Within the NoC, however, the routers and NIs rely on *cycle-level synchronicity* to implement *contention-free routing*, as illustrated in Figure 1.

In our implementation of contention-free routing, the injection of *flow control digits* (flits) is regulated by a TDM table in the NI such that no two flits ever arrive at the same link at the same time. In the figure, there are two IP cores, IP_A and IP_B, that communicate over the network of routers, using the connections c_A and c_B, indicated by a solid and an open-headed arrow, respectively. Connection c_A has slots 0 and 2 reserved in the table, and connection c_B has slot 1 reserved in the table. The TDM table has the same size (or period) throughout the NoC, in this case 4 slots, and

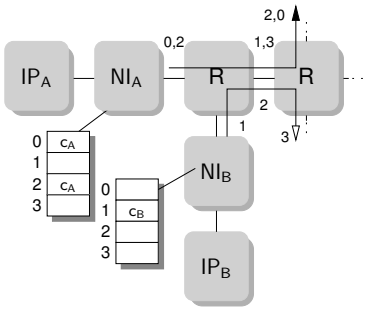


Fig. 1. Contention-free routing.

every slot corresponds to a flit of fixed size, assumed to be 3 words, or *physical digits* (phits), throughout this paper. For every hop along the path, the reservation is shifted one slot, corresponding to the *flit cycle*, i.e. the 3 cycle forwarding delay, of the router. As a result, neighbouring element must be cycle-level synchronous to ensure that the output of one module reaches the next within one cycle, despite any inter-module delay. Note that the logical synchrononicity within the NoC does not limit how the IPs use the network. All interfacing between the IPs and the network uses blocking reads and writes and the network places no assumptions on the production or consumption of data.

We consider it our problem to provide a NoC that: 1) enables large-scale SoCs with multiple applications, 2) allow applications to be developed and verified in isolation, 3) provides real-time guarantees with bounds on latency and throughput to individual applications, and 4) does so with a low cost.

In the following sections we describe how the aelite NoC addresses the aforementioned problems by offering a TDM-based GS-only router (Section IV), skew-insensitive mesochronous link pipeline stages (Section V), and asynchronous wrappers (Section VI). Together, this results in a *flit-synchronous* NoC, with composable and predictable services. We return to the problem statements when discussing the experimental results (Section VII).

IV. Router architecture

The aelite router, depicted in Figure 2, consists of three pipeline stages, corresponding to a *flit size of three words*. The

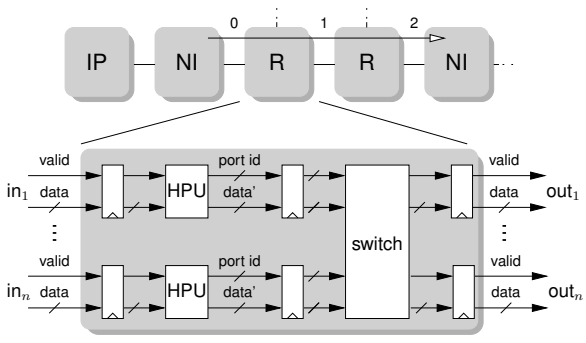


Fig. 2. Router architecture.

first stage synchronises the input data. Thereafter, a Header Parsing Unit (HPU) determines the output port based on the path encoded in the packet header. The selected output port remains the same until an End-of-Packet (EoP) bit is encountered. In contrast to the Æthereal architecture [8], the valid and EoP bits are explicit control signals and do not need any decoding, which *removes the HPU from the critical path*. The output port numbers are one-hot encoded before being fed to the switch which determines the assignment of input to output ports. Thus, three cycles after a flit is presented to a router, the first word appears on the output, as indicated by the open-headed arrow in Figure 2. The aelite router is only parametrisable in its data width and the number of input and output ports (potentially different) and has *no routing table and only a one-word buffer per input port*. It also has *no arbiter* because contention is avoided through off-line scheduling of the flits. This especially benefits asynchronous router implementations [15].

In contrast to VC-based NoCs [3], [4], [6], [7], the router is not negatively affected by the number of connections, service levels, or the real-time requirements of the connections. The aelite architecture is compatible with existing Æthereal design tools, for resource allocation [16] as well as analysis, simulation and synthesis. The benefits over the combined GS and BE Æthereal architecture are: 1) all connections are isolated, thus achieving functional scalability, 2) not having BE *reduces the router to one VC and removes the need for link-level flow control*, which greatly simplifies a mesochronous implementation, thus achieving scalability on the physical level as detailed in Section V, and 3) the hardware implementation is much simpler, thus enabling *lower area and higher speed*, as we will see in Section VII.

A. Limitations

Resources must be reserved in advance and are hence not available to other connections. It is important to note, however, that: 1) the routers are much cheaper than in the original Æthereal architecture, and 2) resource reservations do not have to correspond to the worst-case requirements if this is not needed by the application. In other words, if an application has soft or no real-time requirements there is no need to overallocate resources. Unused resources remain idle (rather than being redistributed [3], [8]) and an attempt to oversubscribe causes the application to slow down due to back pressure. Thus, there is no possibility for an application to violate any contract with the interconnect.

A limitation that the basic aelite router shares with Æthereal is that the NoC requires a *globally synchronous clock* and a *link delay of at most one cycle*. This places strict requirements on the placement of routers and the distribution of a clock. The link delay problem can be mitigated by pipelining links [11], [12]. With the aelite architecture, this is possible by moving the input register, as shown in Figure 2, onto the link itself. However, the clock skew between neighbours must be sufficiently low to avoid sampling in critical regions, severely limiting scalability. This problem is mitigated or completely removed with the introduction of *mesochronous link pipeline stages*.

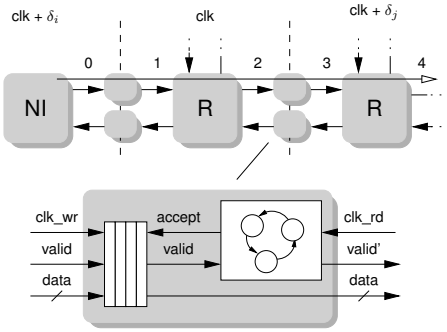


Fig. 3. Link architecture.

V. Mesochronous link

The choice of a link greatly affects how sensitive the NoC is to wire delays and what clock distribution scheme is possible. When the routers and NIs share a clock and thus have the same nominal frequency but different phase relationships, mesochronous links mitigate skew constraints in the clock tree synthesis. Most importantly, mesochronous NoCs are scalable [5], since the phase difference between regions is arbitrary.

Normally, link-level flow control and multiple VCs complicate the implementation of mesochronous (and asynchronous) NoCs [4], [6], [7], [12], [13], due to the increasing amount of control signals required. The latency involved is also reported to limit the maximum achievable operating frequency [5]. In aelite, there is *no need for link-level flow control and only one VC, independent of the number of connections/service levels*. This is a major difference with existing mesochronous/asynchronous NoCs. The challenge in aelite is to provide composable and predictable services without global synchronicity.

To hide the differences in clock phases we do not only put *bi-synchronous FIFOs between neighbouring elements* [4], [13], [17], but also *allocate a time slot for the link traversal*, thus hiding the difference in phase. The architecture of the link pipeline stage consists of a bi-synchronous FIFO [14], [18] and a FSM, as shown in Figure 3. The FIFO adjusts for the differences in phase between the writing and reading clock, where the former is *sourced along with the input data*, thus experiencing roughly the same signal propagation delay [5]. The FSM tracks the receiver's position within the current flit (0, 1 and 2). If the FIFO contains at least one word (valid is high) the cycle a new flit cycle begins (state 0), the FSM keeps the valid signal to the router and the accept signal to the FIFO high during the succeeding flit cycle (3 clock cycles). This is analogous to an *actor* that *fires* in a dataflow graph [19]. Like [4], we assume that the skew between the reading and writing clock is at most half a clock cycle, and that the bi-synchronous FIFO has a forwarding delay less than the number of words in a flit (1-2 cycles) and a nominal rate of

one word per cycle. Under these assumptions, the re-alignment of incoming flits to the reading clock ensures that: 1) flits are presented to the router in their assigned time slot, i.e. not too early and not too late, and that 2) the 3 words of a flit are forwarded to the router in consecutive cycles. The FSM re-aligns incoming flits to the *flit cycles of the reading clock*, thus achieving *flit synchronicity* over a mesochronous link.

With the aforementioned behaviour, the FSM ensures that *it always takes 3 cycles* (in the reading clock domain) for a flit to traverse a link. As illustrated with the long open-headed arrow in Figure 3, this aligns the flit to flit-cycle boundaries, but introduces additional latency. The three cycles, however, are enough to absorb the latency of the FIFO and the skew between the writing and reading clock. Moreover, as the phase difference is guaranteed to be bounded, the FIFO is chosen with sufficient storage capacity to never be full (4 words). The FIFO hence does not need to generate a full/accept signal, and all handshakes are local. Note that in contrast to NoCs that rely on VCs [3], [6], [7], the size and number of FIFOs in the link pipeline stages and the routers is *independent of the number of connections* and does not affect the critical path (and hence scalability) of the NoC. Similar to [5], the mesochronous implementation of aelite has the benefit that the NoC can be conceived as globally synchronous on the flit level. The system designer thus does not need to consider its mesochronous nature. Between neighbouring routers and NIs, the phase difference is limited to half a cycle, but all global constraints are removed. It is also possible to place multiple link pipeline stages in sequence, if required.

A. Limitations

The mesochronous links are only applicable if the entire NoC has the same nominal rate. If the routers and NIs are *plesiochronous* (or even *heterochronous*) [17], then adapting the link is not sufficient. Some router and NIs will be faster than others, and they must be slowed down to guarantee that *input and output is flit-synchronous relative to neighbouring network elements*. This is achieved by introducing *asynchronous wrappers* [10], [20], turning them into *stallable processes* [20].

VI. Asynchronous wrapper

To enable NIs and routers to be plesiochronous (or even heterochronous), we turn the basic aelite router and NI into *stallable processes* [20]. This is accomplished by the *asynchronous wrapper* [10], depicted in Figure 4, that, like a dataflow *actor*, only proceeds from one iteration (flit cycle) to the next once it has synchronised with *all* its neighbours by looking at *the availability of input data and output space* [19]. The wrapper runs synchronously, moving the complexities of clock-domain crossing to the asynchronous links, and consists of Port Interfaces (PI), a Port Interface Controller (PIC), and the router itself, as described in Section IV. Next, we describe the behaviour of the PIs and PIC.

As advocated in [10], each port of the router is managed by a separate PI. We distinguish between Input PIs (IPI) and

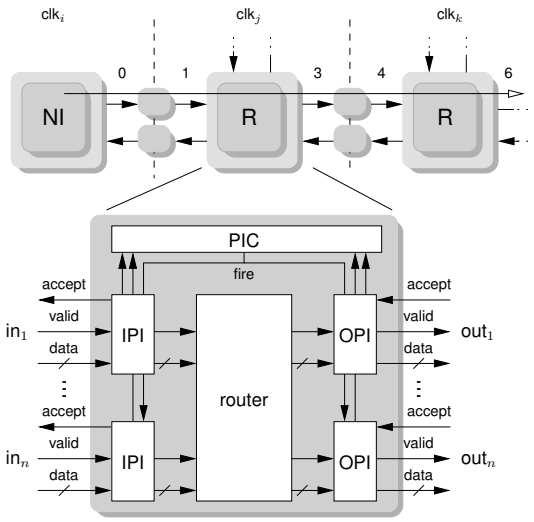


Fig. 4. Wrapper architecture.

Output PIs (OPI). Each IPI and OPI consist of a *synchronous FIFO and an associated counter*. In the IPI, the counter tracks how many words are present in the FIFO. The counter in the OPI, on the other hand, reflects *how much space is not yet reserved*. Therefore, the counter in the OPI is decremented as soon as input data is forwarded to the router, rather than when the data is accepted in the output FIFO. The early reservation ensures that the forwarding delay of the router does not lead to overflow. The IPI and OPI signal the PIC when at least one flit and space for one flit is available, respectively. The flit thus corresponds to *a token* in the dataflow model, and every PI is a *firing rule* [19].

The PIC *fires* once all PIs *fire*. The combined fire signal is fed back to the IPIs where it acts as an accept signal for the input FIFOs. The inputs are then fed to the router and a registered version of the fire signal, with 2 cycles delay (corresponding to the data path in the router without input registers), is distributed to the OPIs as a valid signal for the output FIFOs. The behaviour of the PIC guarantees that flits belonging to the same flit cycle, or tokens belonging to the same *iteration* [19], are passed to the router synchronously. The router responds to the insertion of delays from its neighbours by delaying its inputs and outputs *at the granularity of flits*. Thereby, there is no data stalling inside the router and NI data path. This allows for a simpler implementation of the FSMs in the router and NI as they only have to be stallable before transitioning to a new flit cycle. The delay involved in clock-domain crossing is hidden by adapting the slot allocation as shown with the open-headed arrow in Figure 4.

The functionality of the PIs and PIC ensures correct behaviour as long as all ports are sending and receiving data every flit cycle. There are, however, two additional problems that have to be addressed. First, when there is no useful data to send on a specific output, the router or NI sends *an empty token*, with the only purpose to synchronise with the neighbouring element. Second, a few cycles are spent at reset to

produce initial empty tokens and thus synchronise transmitter and receiver blocks. Otherwise, the system deadlocks.

A. Limitations

Due the flit synchronicity, which is crucial for system-level composable and predictable services, the aelite NoC only runs as fast as the slowest router or NI. This is not a problem in a plesiochronous NoC, which is our current aim. However, to benefit from a heterochronous NoC, additional link-width conversion must be added. Furthermore, the aelite NoC, in its current form, consumes power while idling. The power consumption is reduced by moving to a completely asynchronous implementation [15], or by introducing sleep modes for individual routers. We consider the latter, together with link-width conversion, future work.

VII. Experimental results

To evaluate the aelite design, we return to the problem statements from Section III.

Our first problem, to *enable large-scale SoCs*, is solved by enabling scalability on the physical and architectural level. The physical scalability is achieved by using either the mesochronous links or by adopting the asynchronous wrappers. With the mesochronous links, the constraints on clock skew are local rather than global and an arbitrary (and heterogeneous) number of link pipeline stages can be introduced. These techniques alleviate the designer from strict requirements on clock skew and link delay, thus enabling an effective distributed placement of the NoC components, even with local clock generation. The architectural scalability stems from the ability to add more links, routers and NIs.

The second problem is to *provide functional scalability* by enabling independent development and verification of applications. This is achieved by offering composable services with complete isolation of connections. Additionally, the fabric of the NoC is not negatively affected by the number of connections or service levels.

The third problem concerns *the provision of real-time guarantees*. In aelite, just as Æthereal, the bounds are based on contention-free routing, and thus on the ability to bound the time required for one flit cycle. In this work, we assume that the entire NoC has same nominal frequency[1]. The latency and throughput thus follows directly from the waiting time in the NI (plus the time required to traverse the path), and the fraction of slots reserved, respectively.

The fourth item is *a good performance normalised to cost*. In this work, due to space limitations, we only consider silicon area to be the cost. To determine the area, a number of router instances are synthesised. Results are obtained for worst-case commercial conditions for a 90 nm low-power CMOS technology. Note that all synthesis reported throughout this work are *before place-and-route*, and include *cell area only*. After layout, the area increases and the maximum frequency drops (a utilisation higher than 85% is difficult to achieve and frequency reductions of up to 30% are reported in [12] for a 65 nm technology).

[1]Note that performance analysis of a heterochronous aelite implementation is possible by modelling the links, NIs and routers in a dataflow graph, something we consider future work.

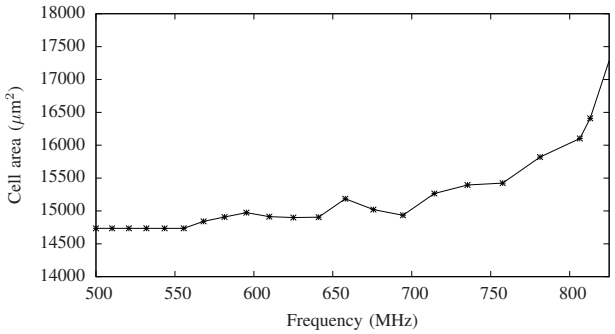
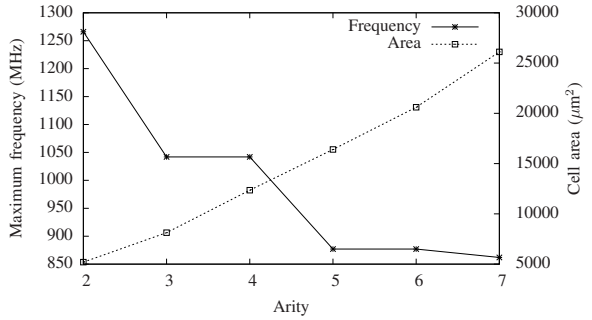


Fig. 5. Frequency and area trade-off.

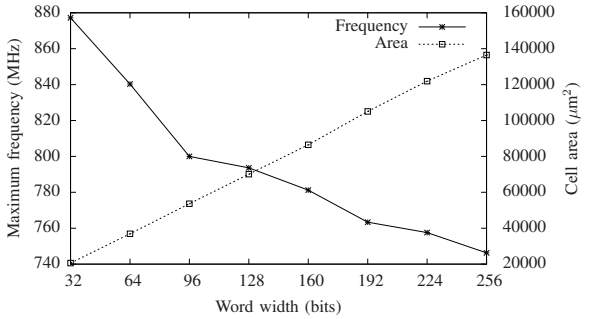
Figure 5 shows the trade-off between target frequency and total area for an arity-5 router with 32-bit data width. As seen in the figure, the router occupies less than 0.015 mm^2 for frequencies up to 650 MHz. The area grows steeply after 750 MHz and saturates around 875 MHz. The results suggest that a frequency of around 600 MHz is a reasonable target after layout and routing. The area and frequency of the aelite router is independent of the number of connections passing through the router, unlike VC-based NoC architectures like [3], [6], [7], and the synthesis results are to be compared with the combined GS and BE router of Æthereal, that occupies 0.13 mm^2 and runs at 500 MHz, when synthesised in a 130 nm CMOS technology [8]. Confirming the observations in [8] and Mango [3], we see that the GS-only NoC architecture provides a much better performance-to-cost trade off than a combined GS and BE NoC. In aelite the difference is roughly 5$\times$ smaller area and 1.5$\times$ the frequency for the same 90 nm technology.

For the mesochronous link pipeline stages, additional cell area is added due to the FSM and bi-synchronous FIFOs. The area of a 4-word FIFO is in the order of 1500 μm^2 when using the custom FIFOs from [18], or roughly 3300μm^2 with the non-custom FIFOs from [4]. For an arity-5 router with mesochronous links the complete router with links is in the order of 0.032 mm^2. This is to be compared to the mesochronous router in [4], or the asynchronous router in [7], that occupy 0.082 mm^2 and 0.12 mm^2 (scaled from 130 nm), respectively. Also note that these two NoCs offer only two service levels and no composability. Extending the combined GS and BE router of Æthereal to mesochronous links is more costly than the aelite GS-only router due to more complex link-level flow control and larger buffers.

Figure 6 shows how the router scales with the arity and the data width, when synthesised for maximum frequency. In Figure 6(a) we see that the area grows roughly linearly with the arity, despite the multiplexer tree in the switch. Figure 6(b) shows how the data width affects the area and obtainable frequency. We observe that the area grows linearly with the word width while the operating frequency is reduced, also with a linear trend. It is clear from our experiments that the aelite



(a) Varying router arity for 32-bit data width.



(b) Varying data width for arity-6 router.

Fig. 6. Total cell area and maximum frequency for varying arity and data width.

router scales to both high arities and wide data widths, thus offering massive amounts of throughput at a low cost, e.g. an arity-6 aelite router offers 64 Gbyte/s at 0.03 mm^2 for a 64-bit data width. Our results also suggest that the aelite router is well suited for more concentrated topologies, with high-arity routers, as proposed in [12].

To demonstrate the ability to provide composable and predictable services (scalability and real-time guarantees) we simulate a NoC with 200 connections, divided across four different applications. The throughput and latency for the connections is randomly chosen, and range from 10 to 500 Mbyte/s and 35 to 500 ns, respectively. With a total of 70 IPs, mapped to a 4 $\times$ 3 mesh with 4 NIs per router (i.e. a concentrated topology). An operating frequency of 500 MHz is sufficient to satisfy the requirements of *all connections* and do so with *no inter-connection interference*. Next, we use the same mapping of IPs to NIs, and keep the same paths through the network, but replace aelite with Æthereal and change the service of all connections from GS to BE. With this configuration, application composability is lost, and the cost of the router network is roughly 5 times as high. For most connections, the average latency observed with BE service is lower than with GS, but the distribution of flit latencies is much larger, and the maximum latencies grow significantly. In fact, the NoC requires an operating frequency of more than 900 MHz before the latency observed during simulation is lower than requested for all connections. From these experiments we can conclude that aelite not only provides temporal isolation that is essential for application composability, but also does so at a low cost and with competitive performance.

VIII. Conclusions and Future Work

A growing number of applications, often with real-time requirements, are integrated on a single System on Chip (SoC). Networks on Chip (NoC) have emerged as a scalable infrastructure for such SoCs, also providing guaranteed services, with latency and throughput bounds. A common approach to providing such bounds, and isolate individual connections, is Time Division Multiplexing (TDM). Using TDM, however, requires a global notion of synchronicity, which is becoming prohibitively expensive with growing chip sizes.

In this paper we present the flit-synchronous *aelite* NoC architecture that, in contrast to existing NoCs, offers scalability on the physical, architectural and functional level. Physical scalability is provided by using mesochronous or asynchronous links, and functional scalability is provided by offering composable services, that isolates independent applications. A complete arity-5 router with mesochronous links requires only 0.032 mm^2 and runs at more than 800 MHz in a 90 nm CMOS technology.

In our future work we aim to extend aelite with link-width conversion and include the asynchronous wrappers in the formal models of the NoC.

References

[1] M. Rutten *et al.*, "Dynamic reconfiguration of streaming graphs on a heterogeneous multiprocessor architecture," *IS&T/SPIE Electron. Imag.*, vol. 5683, 2005.
[2] A. Jantsch, "Models of computation for networks on chip," in *Proc. ACSD*, 2006.
[3] T. Bjerregaard and J. Sparsø, "A scheduling discipline for latency and bandwidth guarantees in asynchronous network-on-chip," in *Proc. ASYNC*, 2005.
[4] I. Miro Panades *et al.*, "A low cost network-on-chip with guaranteed service well suited to the gals approach," in *Proc. NANONET*, 2006.
[5] T. Bjerregaard *et al.*, "A scalable, timing-safe, network-on-chip architecture with an integrated clock distribution method," in *Proc. DATE*, 2007.
[6] D. Rostislav *et al.*, "An asynchronous router for multiple service levels networks on chip," in *Proc. ASYNC*, 2005.
[7] E. Beigne *et al.*, "An asynchronous NOC architecture providing low latency service and its multi-level design framework," in *Proc. ASYNC*, 2005.
[8] K. Goossens *et al.*, "The Æthereal network on chip: Concepts, architectures, and implementations," *IEEE Des. and Test of Comp.*, vol. 22, no. 5, 2005.
[9] M. Millberg *et al.*, "Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip," in *Proc. DATE*, 2004.
[10] J. Muttersbach *et al.*, "Practical design of globally-asynchronous locally-synchronous systems," in *Proc. ASYNC*, 2000.
[11] S. Stergiou *et al.*, "$\times$pipes lite: A synthesis oriented design library for networks on chips," in *Proc. DATE*, 2005.
[12] A. Pullini *et al.*, "Bringing NoCs to 65 nm," *IEEE Micro*, vol. 27, no. 5, 2007.
[13] D. Mangano *et al.*, "Skew insensitive physical links for network on chip," in *Proc. NANONET*, 2006.
[14] I. Miro Panades and A. Greiner, "Bi-synchronous FIFO for synchronous circuit communication well suited for network-on-chip in gals architectures," in *Proc. NOCS*, 2007.
[15] T. Felicijan *et al.*, "Asynchronous tdma networks on chip," Royal Philips Electronics, Tech. Rep., 2007.
[16] A. Hansson *et al.*, "Undisrupted quality-of-service during reconfiguration of multiple applications in networks on chip," in *Proc. DATE*, 2007.
[17] D. Messerschmitt, "Synchronization in digital system design," *IEEE Jour. on Sel. Areas in Comm.*, vol. 8, no. 8, 1990.
[18] P. Wielage *et al.*, "Design and DfT of a high-speed area-efficient embedded asynchronous FIFO," in *Proc. DATE*, 2007.
[19] E. A. Lee and T. M. Parks, "Dataflow Process Networks," *Proc. of the IEEE*, vol. 83, no. 5, 1995.
[20] L. Carloni *et al.*, "Theory of latency-insensitive design," *IEEE Trans. on CAD of Int. Circ. and Syst.*, 2001.