The Aethereal Network on Chip after Ten Years: Goals, Evolution, Lessons, and Future

(Invited Paper)

Kees Goossens
Eindhoven University of Technology
k.g.w.goossens@tue.nl

Andreas Hansson
University of Twente
a.hansson@utwente.nl

ABSTRACT

The goals for the Æthereal network on silicon, as it was then called, were set in 2000 and its concepts were defined early 2001. Ten years on, what has been achieved? Did we meet the goals, and what is left of the concepts? In this paper we answer those questions, and evaluate different implementations, based on a new performance:cost analysis. We discuss and reflect on our experiences, and conclude with open issues and future directions.

Categories and Subject Descriptors

B.7 [**Hardware**]: Intregrated Circuits

General Terms

Performance

Keywords

Network on chip, rate control, circuit switching

1. APPLICATION DOMAIN AND GOALS

Work on the Æthereal network on chip (NOC) started at Philips Research, for systems on chip (SOC) in the consumer-electronics (CE) domain, in particular digital TV (DTV) and set-top boxes (STB). These systems include applications such as audio and video decoding and improvement that have *real-time requirements* and *high computational performance at low cost* (high bandwidth (Gb/s) to area (mm^2) ratio). Moreover, unlike for general-purpose computing, consumers do not tolerate CE devices misbehaving or crashing. Hence CE SOCs have to be *robust*, in the sense that a faulty IP or application, must not cause the entire system to break.

DTV and STB SOC architectures were characterised by the use of a single external SDRAM for all inter-IP shared-memory communication [5]. All IPs, most of which were hardware accelerators, had dedicated wires to the SDRAM, leading to many global wires and wire congestion at the SDRAM interface. IPs required a guaranteed minimum bandwidth and maximum latency to ensure an end-to-end SOC bandwidth and latency for video and audio processing. In contrast, the CPU's average latency and bandwidth requirements were strict, but not real time.

With this background, several trends were emerging in 2000, which we took as our problem statement:

1. Physical, back-end problems, in particular difficult timing closure and congestion of many long global wires, introduction of multiple synchronous regions and GALS.

2. Logical, scalability problems, in particular tri-state and broadcast busses with a single global arbiter, and bottlenecks in the SOC architecture such as a single shared SDRAM for all inter-IP communication.

3. Increasing cost of designing; integration of working components often did not lead to a working system, both at the physical level (e.g. resizing bus tri-state buffer strenghts) and at the logical level (e.g. resizing depths of decoupling FIFOs between IPs). Moreover, analysing or determining the real-time performance of SOCs was time-consuming and complex. Finally, post-silicon SOC validation and debug were taking an increasing percentage of the design cycle.

Note that low power was absent from the list of requirements, as DTV and STB applications are tethered.

2. CONCEPTS

We discuss Æthereal's concepts in turn. We use a multi-hop interconnect to solve back-end and scalability problems. Next, real-time applications and the cost of designing are addressed by *guaranteed performance* with a corresponding *notion of time.* The consumer-electronics domain requires low cost, for which we introduced *best-effort traffic.*

2.1 Guaranteed Performance

To offer real-time performance (guaranteed service, GS), e.g. a maximum latency or a guaranteed minimum bandwidth over a finite interval, resource budgets must be reserved and enforced. A connection with its associated minimum bandwidth and maximum latency is the NOC equivalent of a resource budget. Real-time NOCs have essentially two options: non-blocking routers with rate control [16], and (virtual)-circuit switching.

Rate Control. Non-blocking routers with a rate-controlled service discipline [16] are the first way to achieve real-time guarantees. Routers are non-blocking, i.e. when packets arrive at a router, they can be routed directly to the appropriate output links without switching conflicts on the crossbar. Queueing occurs only at the output ports of the router. With these assumptions, *only links are arbitrated*, and a connection can be modelled as traversing *a number of queueing servers*, with each server modeling the output link of a router. Many different arbitration policies can be used, with different characteristics in terms of latency, buffer sizes, utilisation, fair use of slack, etc. [16]. Note, however, that priorities alone offer guarantees to only one circuit, and controlling the rate is essential. Traditionally, N^2 output buffers and a $N \times N^2$ crossbar are used. Alternatively, N output buffers with a $N \times N$ crossbar can be used, with both running at N times the link frequency. But this is not feasible in NOCs, where the router and link speeds are the same. Mango [2] is a rate-controlled NOC, but uses virtual-circuit buffering rather than the traditional output buffering.

Note that rate-controlled service disciplines require that all routers act as independent servers. A packet cannot, therefore, block a link (or router crossbar) for other packets. As a result, either store-and-forward or virtual-cut-through switching must be used. Or, if *virtual channels* are used for guaranteed services, as is often proposed, *as many channels as circuits* must be used, in addition to a non-blocking crossbar. This then coincides exactly with Mango's virtual-circuit buffering with a non-blocking crossbar.

Circuit Switching. SOCBus [15] was the first pure circuit-switching NOC. No resources, i.e. wires between and inside routers and any (pipeline) buffers, are shared between connections. Once a connection has been established, real-time guarantees are trivially achieved. However, pure circuit switching does not reduce the number of global wires.

Traditionally, *frequency multiplexing (FDM)* and/or *time multiplexing (TDM)* have been used to address this. Note that even with FDM or TDM data never contend for resources in the network: they never wait and no buffers are required. Optical NOCs [3] use FDM, and Æthereal and Nostrum [10] use TDM. Since the flight time of photons is neglible and because they are hard to buffer, FDM circuit-switching NOCs are not pipelined.

In TDM NOCs, however, pipeline buffers on links or in routers are essential to ensure high operating speeds. Hence a connection does not use the same time slot for its entire path; instead it increments at every hop. Figure 1 illustrates how multiple connections are mapped on a TDM NOC. Since only one datum is guaranteed to arrive in every time slot at every input, and since it leaves in the next time slot, routers only require a pipeline buffer at each input. The table in each router indicates to which input each output is connected, for each slot (allowing multicast.) Since circuit switching is used and routers know where to switch incoming data to, guaranteed-service (GS) data required no routing headers. Later versions of Æthereal, described below, employ *virtual*-circuit switching where packets with routing headers tell routers where they should go.

Thus links are shared (reducing the number of global wires), and routers are small (N minimal input buffers, minimal $N \times N$ crossbar). Routers are also fast: no arbitration is required, as contention is absent since all communication is statically scheduled. But TDM requires that all routers have a global notion of time, to keep their TDM slots aligned.

2.2 Notion of Time

TDM (and FDM) (virtual)-circuit switching treat the entire NOC as a *single shared resource with a single arbiter.* In other words, packets wait only at the ingress network interfaces (NI) until their TDM slot, after which they progress without contention to the egress NIs, with minimal latency. This differs from rate-controlled (and other) NOCs, where each router independently arbitrates incoming packets. Arbiters at each router in a packet's path are not aligned, and a packet may incur the worst-case delay at every hop.

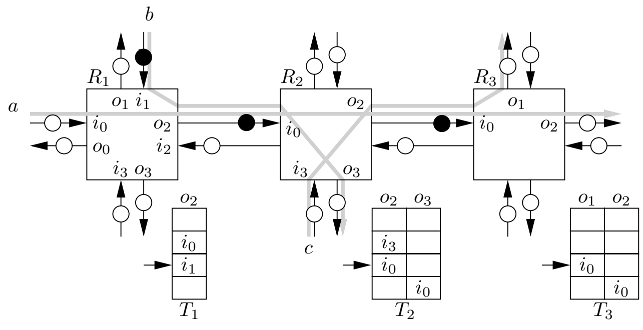

Figure 1: Contention-free routing.

Æthereal has a global notion of time because all routers are always in the same TDM slot. Our main innovation was to implement the *single global arbiter and the global notion of time in a distributed manner.* The NOC is defined as a dataflow graph, where routers are single-rate actors. On reset, all routers produce a token on all outputs. Following this, in an infinite loop they all read a token on all inputs, increment the slot, and switch each token to an output according to the switching table. (Note the absence of contention.) Tokens are either empty or full (corresponding to a circuit datum, or a virtual-circuit packet). This model has been implemented synchronously, mesochronously [8], and asynchronously [4]. In fact, since no contention occurs, the asynchronous router requires a synchroniser to wait for all incoming tokens, but no asynchronous arbiter for the crossbar. Since all routers only advance to the next slot when their neighbours have, the NOC is logically synchronous and runs at the speed of the slowest router.

This model of time is suitable for analytical performance analysis, as well as allocation/synthesis, i.e. computing paths, slots, and buffer sizes for given communication requirements (bandwidth and latency, use cases), see Section 3.4

2.3 Low Cost and Best Effort

Minimal area cost was a foremost concern from the outset. Although this was achieved by TDM circuit switching, we made the classic mistake of *confusing utilisation with efficiency.* TDM is not work conserving, i.e. slots may be empty even though data is waiting in the NIs. We therefore included lower-priority best-effort (BE) traffic in the NOC that used the unallocated and unused slots. The NOC logically consisted of a GS NOC and a BE NOC (input buffers, round-robin arbitration) that only shared the links. However, BE traffic increased the GS flit size of one word to the flit size (and latency) of three words for BE. However, utilisation is not relevant, but the performance:cost ratio is. And the addition of BE traffic significantly worsened the latter.

Since RAM or flip-flop based BE buffers occupied up to 80% of the router, we designed a dedicated hardware FIFO [14] that reversed this ratio. In the NIs it implemented the (virtual)-circuit buffers, and also served as the clock-domain crossing between IP and NOC clock domains.

2.4 Routing, Flow Control and Deadlock

The NIs were strictly decoupled in a kernel and a shell, corresponding to the network and transport layers of the protocol stack. Shells serialise distributed-shared-memory protocols, such as AXI, to a streaming protocol accepted by the kernel, which may also be used directly by IPs.

Input buffering is used for both GS (one phyt) and BE (a number of flits) in routers, which is independent from the number of circuits passing through. (Virtual)-circuit buffering is used in both ingress and egress NIs, i.e. a request and response buffer per connection.

Absence of contention for GS traffic entails that no link-level flow control is required, but end-to-end (NI-NI) flow control per connection is essential. We opted for credit-based flow control because it results in smaller buffers than other schemes, although it requires higher bandwidth. Routing deadlock does not occur due to absence of contention. End-to-end flow control avoids all deadlocks, including message and higher-protocol deadlocks; we therefore have a combined request/response NOC [7]. Stalled IPs cannot, therefore, negatively impact other IPs, increasing robustness. BE traffic uses link-level flow control, and is also connection-based to avoid all deadlocks. Special BE setup and teardown packets were used to create and remove connections, concurrently and pipelined from any NI.

In conclusion, the need for real-time performance led to TDM circuit switching; a focus on low cost led to the inclusion of best-effort traffic; and the robustness requirement led to use of end-to-end flow control. Our main innovation was the use of a single global TDM arbiter that was implemented through distributed handshaking between routers.

3. EVOLUTION AND CURRENT STATUS

Given the basic concepts, a decade of continuing research resulted in a multi-processor NOC-based platform called CompSOC [6] with an accompanying design flow. We first discuss the architecture changes, then the design flow.

3.1 Use Cases and Composability

Robustness was one of the goals of Æthereal, which translated in GS connections and end-to-end flow control. Quite soon, however, the concept of *use case, a set of concurrently running applications*, became important. In many SOCs, multiple applications can run together, and can be switched on or off dynamically, often under user control. Use cases have two important repercussions. First, assuming that all applications are always active is unduly pessimistic and expensive. Hence the mapping of connections on the NOC should take applications and use cases into account.

Second, the notion of robustness was refined to *composability, i.e. absence of interference between applications.* Within an application, tasks exchange data and are hence dependendent on each other. Between applications, however, interference should be avoided. Only in this way can they 1) be developed, verified, and debugged independently (e.g. by independent software vendors, ISV), and 2) can they be integrated in a larger system without any unexpected side effects to application or system. Our notion of composability separates the resource sharing (scheduling) within applications, which may be real-time or not, from that between applications, which must be independent. This refines Kopetz's time-triggered approach [9], and enables dynamic scheduling and the use of slack within applications.

In essence, use cases and composability both require a *virtual platform per application.* Users of resources, including tasks on processors, connections in a NOC, and buffers in memories, are given a resource budget. A virtual platform is the collection of resources and budgets of an application (a set of tasks, connections, and buffers). For a predictable (real-time) platform, only minimum budgets have to be defined. For a composable platform, the budgets must be constant, and the times they are handed out to an application are independent of other applications. Intuitively, using TDM on all resources (independently) is the simplest way to achieve this: slots allocated to an application are fixed and independent of others, and unallocated or unused slots go to waste. Although Æthereal's GS connections are therefore composable, memories (especially SDRAM) [1], and processors require more work.

3.2 Best Effort, Low Latency, and Cost

Our approach for GS did not change. However, the inclusion of BE traffic was a mistake, and it is no longer (by default) used in the NOC for three reasons. 1) (our) *best effort does not offer low latency* in a loaded NOC; 2) the *performance:cost ratio is much worse than for GS traffic*; 3) use of *BE quickly breaks composability* between applications.

GS traffic such as video streams, tends to have strict minimum bandwidth (and jitter) constraints, but is relatively latency tolerant. Processor traffic, especially cache misses, on the other hand, is latency critical, with a focus on average rather than minimum latency and bandwidth. Intuitively, low-latency traffic should use BE connections. Unfortunately, this is not the case, because BE traffic has a lower, not higher, priority than GS traffic. However, perhaps more important, the performance of BE depends strongly on the NOC xload and slack. Since in DTV and STB applications GS reserves and uses up to 60% of the NOC capacity, BE performs rather poorly and does not offer a low latency.

GS vs. GS+BE Performance:Cost Trade-Off

Consider the raw performance:cost ratio of a 8x8 GS+BE router with 4-flit BE buffers of 0.6 GHz / 0.07 mm^2 (65 nm) versus that of a GS-only router of 1.9 GHz : 0.022 mm^2. The difference is a factor of 10, which allows us to replace, at no additional cost, a BE connection with a given required average bandwidth by a GS connection guaranteeing 10 times the required bandwidth. Only if the difference between average and worst-case required bandwidth is more than a factor of 10, or if statistical multiplexing between different connections can be relied upon, then perhaps a case can be made for BE. In this simple example we have not taken into account effects such as (absence of) contention, or the maximum load of a BE NOC (around 50%) versus that of a GS NOC (closer to 100%). Figure 2(a) shows a frequency:area:power trade-off for 5x5 GS routers and GS+BE routers (with input buffers of 4 or 8 flits), both with and without clock gating, for a 65 nm low voltage library. Similarly, GS NI and GS+BE NI with 16 TDM slots and 8 ports figures (without connection buffers) are shown. (We use frequency rather than raw link bandwidth as performance metric to more easily compare routers and NIs. In Figure 2(b), described in Section 4.2, we extend the experiment to nett bandwidth.) Since GS routers contain little buffering, clock gating makes not much of a difference. GS+BE routers do contain much buffering (additional 5x4x3 or 5x8x3 words for BE), and they are at least twice as large and power consuming as GS routers. All buffers are based on registers, and use of our hardware FIFOs will reduce both area and power, especially for GS+BE components. In all cases the clock uses between 14 and 22 percent of the power.

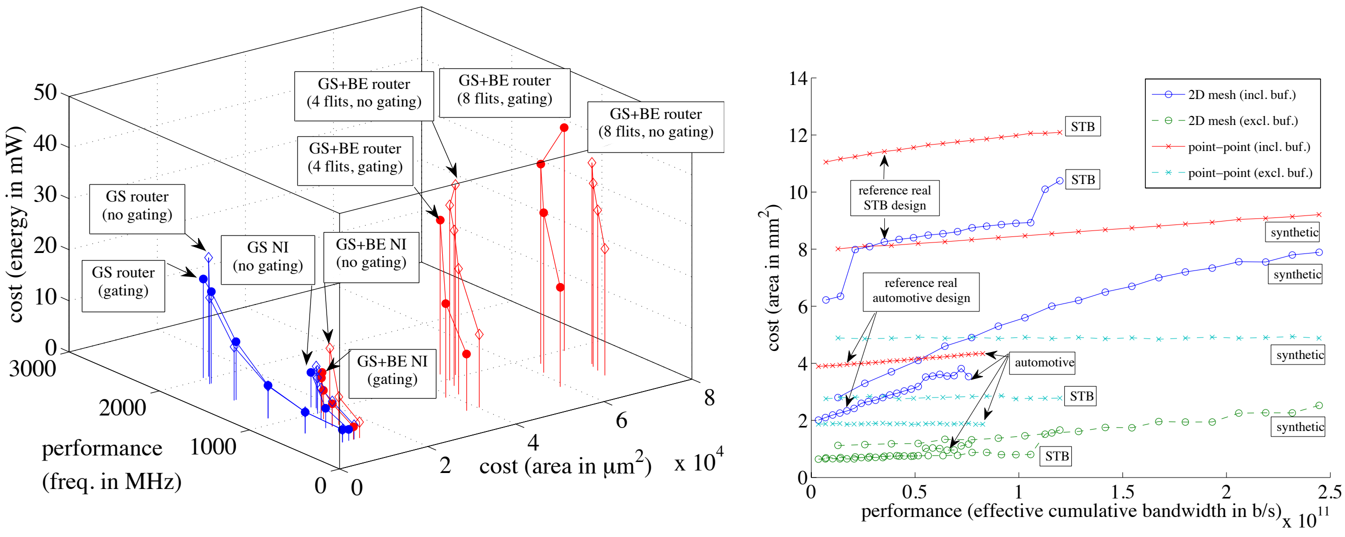

Figure 2: Left: performance:area:energy trade-off for router and NI. Right: performance:cost trade-off for GS NOCs.

Best Effort and Composability
Assume two applications, one best-effort by independent software vendor (ISV) one, and another real-time application by another ISV. BE traffic uses the unallocated or unused slots in the NOC. The unused GS slots belong to a different application the one that the BE connection belongs to. Hence the BE application is influenced by the absence or presence of the real-time application. The BE connection (application) receives more or less bandwidth depending on the use of the NOC by the GS connection (application). Note that extra slots can cause an application to miss its deadline if it is not performance monotonic, through scheduling anomalies. At a more mundane level, it complicates debugging the BE application, because its behaviour depends on the other real-time application, the sources or even executable of which the ISV may not have at its disposal, because it is supplied by another ISV.

Related to the removal of BE traffic, we moved to virtual-circuit switching using packets, which allows deletion of the slot tables from routers, making them much smaller. Arbitrary topologies can be used, without routing restrictions. To set up and tear down connections, only NIs are now programmed, using memory-mapped IO on GS connections. Adding or removing connections is composable, i.e. does not affect other active connections.

3.3 Protocol Stack, Busses, and Clocking

Although the division of the NI in the transport-level shell and network-level kernel was good first start, we further split the shell in a local bus, a simpler shell, and a clock-domain crossing block. The local master bus essentially demultiplexes distributed-shared-memory requests, and enforces the right order on the responses. The shared-memory requests and responses are (de)serialised to a streaming protocol by shells. Slave busses are similar, except that they multiplex incoming requests, and hence require arbitration. Our design flow, described below, generates the hardware, but also computes the configuration (i.e. address maps, arbiter settings) for all those components, based on end-to-end (i.e. master-bus-NOC-bus-slave) requirements.

Regarding clocking, busses and shells operate on the IP clock, and they are connected by clock-domain-crossing blocks; routers operate mesochronously on the NOC clock [8].

3.4 Design Flow

The NOC architecture is only half the work: during the past ten years most of the effort on Æthereal was spent on the development of a design flow. Based on the specification of IP blocks and their (multi-use-case) communication requirements, it automates the following tasks:

1. Dimensioning and instantiation of hardware, i.e. generating a NOC topology, and then (constrained) optimising of the NIs, local busses and their decoders and arbiters, and binding of IPs to these, as well as the sizes of buffers in NIs.

2. For all use cases, computing the run-time-programmable *configuration* of hardware, i.e. paths, TDM slots, address maps for master busses, and arbiter settings for slave busses.

3. Generating drivers to (re)program the NOC with a configuration, at run time, and with real-time constraints. Applications (i.e. their connections) can be composably started and stopped independently at run time.

4. Generation of TLM SystemC models and RTL implementation, including *traffic generators, testbench, and performance monitors.*

5. Automatic inclusion of a *test & debug infrastructure*, e.g. distributed monitors, event propagation for cross-triggering, and transaction-based stepping/stopping control [12].

Our design flow is fully automated for ASIC and FPGA (synthesis, compilation, loading, etc.). It is unique in allowing constraints on both bandwidth *and latency*, from which *NI buffer sizes* and *arbiter settings* for shared-slave busses are computed. Performances is guaranteed for connections (end to end: master-NOC-slave-NOC-master), and even for entire applications running on CompSOC platform, using dataflow modeling [11, 6].

4. REFLECTIONS AND LESSONS

In this section we reflect on our experiences; in particular, “selling points,” scalability, TDM, and admission control.

4.1 (Unique) Selling Points

Æthereal was conceived for the DTV and STB domain, where, ironically, it turned out to be the hardest to compete with the incumbent architecture [5]. An interconnect with a single slave (i.e. the external SDRAM controller) for many masters (CPU, VLIWs, and many hardware accelerators) is naturally a circuit-switching tree with a single arbiter at the root. However, in the end, a NOC was found to be superior for several reasons. First, with the concept of use cases, applications could be switched on and off individually. Previously, each combination of applications would be modelled as a separate application of which there were hundreds, and which required a global reset to start or stop. Second, the use of a protocol stack allowed more efficient (serialised and pipelined) link-level protocols, which reduced the number of long global wires. This alleviated the SOC back-end problems such as timing closure. Third, since the circuit switching operated at the granularity of large SDRAM transactions the NIs required much distributed buffering, leading to many SRAM instances. By using a NOC, the interconnect transport granularity was reduced to flits. Hence buffering of SDRAM bursts could be concentrated at the SDRAM interface, which allowed the use of fewer, larger (and hence more area efficient) on-chip SRAMs. Finally, the SOC architecture started to include multiple slaves, e.g. multiple SDRAM interfaces and chip-to-chip links. Rather than duplicate multiple tree-like interconnects, a NOC naturally allowed balancing of traffic over multiple SDRAMs, either within or between use cases.

NXP's automotive infotainment SOCs had very different architectures, with multiple embedded memories, and multiple data and control busses with bridges. These were a natural candidate for a replacement by a NOC, although back-end problems were not as severe as for the DTV and STB SOCs. Instead composability was the compelling reason for our NOC, since multiple applications from independent (software) vendors were integrated in a single SOC. Composability aims to ease application integration, and ensure application/SOC robustness and stability, which are essential in the automotive domain.

In both cases, it is not feasible to introduce composability or predictability in the entire SOC in one go. However, since the NOC integrates the IPs, composability can already be achieved if they are not shared by different applications. This is usually the case for hardware accelerators, and often feasible for processors (especially DSPs and VLIWs). Only the off-chip SDRAM is almost always shared, and must be made composable and predictable too [1].

4.2 Scalability

NOCs are claimed to be scalable, which here we define as: the *performance:cost ratio of the NOC is constant*, i.e. cost depends linearly on the requirements on bandwidth and latency. This encompasses adding slaves, increasing the offered load, and to some extent the *diversity in requirements* (ratios of bandwidths) of different connections. The performance:cost ratio of a GS NOC versus a GS+BE NOC was discussed in Section 3.2. Here we concentrate on the scalability of the GS NOC.

First, observe that our GS routers are independent, in terms of performance (frequency) and cost (area, notably buffers), of the number of connections passing through them. This contrasts with rate-controlled routers, where virtual-circuit buffering or output buffers are used. The size of the latter depends on the frame size of the guarantees, which is comparable to the slot table size, and indirectly the number of connections (more on this below). As mentioned previously, this is not the case for NIs, which use virtual-circuit buffering (also to avoid deadlock, with end-to-end flow control). Hence the total area cost of all NIs depends on the number of connections. Their performance does not, however, since the connection buffers are not in the critical path. The total area cost of a NOC hence depends on the topology (i.e. the number of routers, and packetisation and scheduling parts of NIs) plus a (NI buffering) part that depends linearly on the number of connections. (A minor complication is that each NI requires a connection for programming.)

Assuming that NI buffers are dimensioned for maximum performance (cf. end-to-end flow control), and that NOC clients always accept data as soon as it is offered, the bandwidth of a GS NOC scales linearly with its number of routers, as there is no contention. However, this supposes that all TDM slots are used, which depends on the IP port to NI binding and on the NOC topology. The worst performance (equal to that of a single link) occurs when all connections are routed over a single link in the NOC. Conversely, the best performance (the sum of all links) is obtained when each connection, consisting of two IP ports, is implemented with a dedicated source NI and destination NI, connected without intervening routers. These two extremes also illustrate that the number of TDM slots reflect the (design-time) contention of the given use case and topology, i.e. more slots are required when more (diverse) connections share a link. The GS NOC saturates at 100% load, unlike a BE NOC which effectively saturates much earlier.

Scalability Experiment

The scalability of (any) NOC depends to a large extent on the scalability of the required traffic: if all traffic converges on a single slave, then no NOC will scale. The reference use case used here is a reasonable intermediate, and contains 70 GS connections, with 32 masters and 32 slaves. 10% of the connections have a maximum con/divergence degree of 10 (i.e. creating bottlenecks); the rest has a maximum degree of 3. Each connection has a burst size between 8 and 32 bytes, and a random *nett required bandwidth* between 35 and 205 MB/s. cumulative total of 8046 MB/s is available on the AXI write data groups. The design flow computes the required raw bandwidth requirements (which is 24 GB/s in total), taking into account AXI byte masks and addresses, and NOC packetisation and end-to-end flow control credits. To vary the requirements we uniformly scale the bandwidths of all connections between 0.2 and 3.8 times the reference, with a maximum cumulative nett required bandwidth of 30 GB/s. Given the use case, our design flow finds the smallest NOC of a given topology (point-to-point or mesh), and a valid configuration (paths, slots), and with the required buffers. Real requirements for a set-top box SOC and automotive infotainment SOC are similarly scaled (0.2-3.4 and 0.2-4.4). Cost is the area (estimated by the design flow) in 90 nm of a NOC running at 500 MHz with 32 TDM slots, and includes local busses, NIs, and routers. Importantly, it includes the additional capacity (routers, buffers, TDM slots) required due to less-than-perfect mapping, routing, and slot allocation, but also more slots for low latency, etc.

Figure 2(b) shows the cost of a NOC as we increase the required performance. It illustrates that sharing wires is beneficial, as the mesh topology is significantly cheaper that the point-to-point topology. However, they converge under heavy loads, as links are progressively more shared (making NI buffers larger), and/or requiring more links and routers. The former effect dominates, as shown by the increasing percentage of buffer area. For both topologies, however, the performance:cost ratio is remarkably constant, *showing that our GS NOC is indeed scalable*. The use of static source routing or TDM is no impediment to scalability.

4.3 Time-Division Multiplexing

Our choice of TDM has often been contested, for several reasons. First, TDM inversely couples latency and rate, i.e. a low latency is obtained only at the expense of a high bandwidth. Hence, in essence, low latency means more (less-shared) wires, rather than fewer wires shared with priority-based rate-control, such as Mango [2]. Second, TDM is interpreted to imply a synchronous design. As discussed previously, mesochronous and asynchronous implementations are also possible, with the limitation that slot tokens keep part of routers or NIs alive, even when no data is sent.

In our opinion these disadvantages are more than offset by the advantages, namely: for any number of slots TDM is cheap and fast, i.e. has no arbitration in the routers, and NI arbitration can be pipelined. It also attains 100% maximum effective bandwidth, and the restrictions on the distribution of budgets it can offer are less severe (limited by the number of slots vs. by the number of priorities and precision of rates). Most important, however, is that TDM is composable, both in steady state, and during (re)configuration: connections that are not reconfigured do not experience (transient) interference. (Rate-controlled) priority-based approaches fail on all these points. Note that TDM is overly composable, in the sense that it also disallows the use of slack between connections of the same application. Ideally a two-level arbitration scheme is used; TDM between applications and a work-conserving scheduler within an application. However, this requires at least a virtual-circuit buffer per application [13] and a more complex arbiter.

4.4 Admission Control

Admission control, or the complexity of checking whether a connection can be accommodated on a given NOC, varies for NOCs that offer guaranteed services. For pure circuit switching it only requires computing a free path between source and destination. For unbuffered frequency-division-multiplexing (FDM) circuit switching, admission control requires both (a single) frequency assignment and routing from source to destination, which is more complex. For TDM NOCs, the complexity is higher yet, as successive links on the path require consecutive slots. In our platform connection admission control is performed at design time, when multiple use cases are mapped. At run-time admission control takes place at the level of entire applications.

5. OPEN ISSUES AND FUTURE DIRECTIONS

Although Æthereal achieved its goals, several new trends can be discerned: for hardware technology, changing performance:cost trade-offs, and cost of designing. Below, we discuss their impact.

1) Increasing *variability and unreliability* in existing silicon technologies can be addressed by NOCs in much the same way general computer networks have, through protocol layering and gross overdimensioning. Even then, the Asynchronous Transfer Mode (ATM) network, an inspiration for Æthereal, found guaranteed services hard to achieve. In the medium term, an asynchronous Æthereal will suffice (note that only the hardware changes, the design flow not). In the long run, building real-time systems from heterogeneous, and worse, unstable components will be hard.

2) Emerging new technologies, in particular *optical and wireless NOCs, and 3D die stacking*, open up the SOC and NOC design space. Optical NOCs and wireless (within package) communication are interesting additions to, rather than replacements of, conventional NOCs. Hierarchical and heterogeneous NOCs pose challenges for end-to-end performance guarantees. 3D stacking per se does not necessarily impact a NOC much beyond the link level. But its effects on the SOC architecture will reflect back on the NOC requirements, especially due to new memory hierarchy options.

3) With the claimed emergence of "dark silicon," i.e. not all silicon can be used simultaneously, the *performance:cost trade-offs change*. It is then effective to over-dimension and operate at lower frequencies. In theory, multiple parallel, simpler components work better than fewer complex components. A GS NOC fits the bill, except that in the extreme case, components are not even shared, leading to pure circuit switching. But it remains to be seen if the abundance of transistors materialises, and if it also holds for global wires.

4) The cost of designing ASICs *and* FPGA-based systems is rising unabatedly. We must ease programming with multiple application-specific programming models with platform support (note that we advocate *software* coherency for *selected* applications), and with *divide-and-conquer design approaches such as composability* (virtualisation, including of performance). For variability run-time support for test, calibration, and post-silicon debug is required [12].

Acknowledgements Many people contributed to Æthereal. We thank them all, especially Jef van Meerbergen.

6. REFERENCES

[1] B. Akesson, et al. Composable resource sharing based on latency-rate servers. In *DSD*, 2009.

[2] T. Bjerregaard, et al. Scheduling discipline for latency and bandwidth guarantees in asynchronous NOC. In *ASYNC*, 2005.

[3] L. P. Carloni, et al. NOCs in emerging interconnect paradigms: Advantages and challenges. In *NOCS*, 2009.

[4] T. Felicijan, et al. Asynchronous TDMA networks on chip. Technical Note 2004/00801, Philips Research, Jan. 2004.

[5] K. Goossens, et al. Interconnect and memory organization in SOCs for advanced set-top boxes and TV. In *Interconnect Centric Design for Advanced SoC and NoC*. Kluwer, 2004.

[6] A. Hansson, et al. CoMPSoC: A template for composable and predictable multi-processor system on chips. *TODAES*, 2009.

[7] A. Hansson, et al. Avoiding message-dependent deadlock in network-based systems on chip. VLSI *Design*, May 2007. Hindawi

[8] A. Hansson, et al. aelite: A flit-synchronous network on chip with composable and predictable services. In *DATE*, 2009.

[9] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer, 1997.

[10] M. Millberg, et al. Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip. In *DATE*, 2004.

[11] P. Poplavko, et al. Task-level timing models for guaranteed performance in multiprocessor NOCs. In *CASES*, 2003.

[12] B. Vermeulen et al. Debugging distributed-shared-memory communication at multiple granularities in NOCs. *NOCS* 2008.

[13] W.-D. Weber, et al. A quality-of-service mechanism for interconnection networks in system-on-chips. In *DATE*, 2005.

[14] P. Wielage, et al. Design and DFT of a high-speed area-efficient embedded asynchronous FIFO. In *DATE*, 2007.

[15] D. Wiklund, et al. Socbus: switched network on chip for hard real time embedded systems. In *IPDPS*, 2003.

[16] H. Zhang. Service disciplines for guaranteed performance service in packet-switching networks. *Proc. IEEE*, Oct. 1995.