A Design Flow for Application-Specific Networks on Chip with Guaranteed Performance to Accelerate SOC Design and Verification

Kees Goossens, John Dielissen, Om Prakash Gangwal, Santiago Gonzalez Pestana, Andrei Radulescu, Edwin Rijpkema

A Design Flow for Application-Specific Networks on Chip with Guaranteed Performance to Accelerate SOC Design and Verification

Kees Goossens, John Dielissen, Om Prakash Gangwal, Santiago González Pestana,
Andrei Rădulescu, and Edwin Rijpkema
Philips Research Laboratories, Eindhoven, The Netherlands

Abstract

*Systems on chip (*SOC*) are composed of intellectual property blocks (*IP*) and interconnect. While mature tooling exists to design the former, tooling for interconnect design is still a research area. In this paper we describe an operational design flow that generates and configures application-specific network on chip (*NOC*) instances, given application communication requirements. The* NOC *can be simulated in SystemC and* RTL VHDL*. An independent performance verification tool verifies analytically that the* NOC *instance (hardware) and its configuration (software) together meet the application performance requirements. The Æthereal* NOC*'s guaranteed performance is essential to replace time-consuming simulation by fast analytical performance validation. As a result, application-specific* NOC*s that are guaranteed to meet the application's communication requirements are generated and verified in minutes, reducing the number of design iterations. A realistic* MPEG SOC *example substantiates our claims.*

1 Introduction

A SOC is naturally composed of computation and storage elements (intellectual property blocks or IP) that are interconnected by communication elements (busses, networks on chip or NOC). In this paper, we focus on NOC interconnects because of their modularity, scalability, and other advantages for large SOCs.

Mature tooling exists to design individual IP, such as RTL synthesis and processor synthesis. Moreover, extensive IP re-use (of memories, processors, and application-specific blocks) is common practice. In contrast, an interconnect is specific to a SOC because the communication requirements depend on the composition of IP, which is application specific. Its design costs cannot be amortised over multiple SOCs, because it cannot be re-used whole. *Tools for* NOC *synthesis are therefore essential for fast and efficient* SOC *design.* These tools depend on the modularity of NOCs; i.e. (application-specific) NOCs are composed of two re-usable parametrised components: routers and network interfaces (NI).

In this paper we describe our design flow to dimension and generate application-specific NOC instances, given the communication requirements of the application. The NOC hardware (router and NI topology), and the IP port to NI port mapping are described in XML, which are translated to synthesisable RTL VHDL, and to SystemC. Minimum buffer sizes can also be computed. Every NOC instance is programmable, and its configuration (software) is generated in XML format for SystemC and VHDL simulation, and in C format for embedded processors in the SOC. VHDL simulation is bit and cycle-accurate, and SystemC transaction-level simulation is flit-accurate. If IP are not yet available for simulation, traffic generators are used that mimic their communication behaviour, as specified in the application communication requirements. A powerful new element in our design flow is performance verification, described below.

Impact of guaranteed NOC services on design flows

One of the major challenges in SOC design is *ensuring that the* SOC *fulfills the (real-time) application requirements under all circumstances*, such as video throughput and latency for set-top boxes, or packet loss and throughput for network processors. Assuming the IPs have the right performance (operations per second, storage capacity, etc.), we must generate a NOC with the right performance. We will show that using a NOC with guaranteed services (such as minimum throughput, maximum latency and jitter, etc.) as opposed to a best-effort NOC has important benefits for a design flow. In particular, this results in a fundamental difference in how performance is validated.

Using a NOC *with best-effort services*, any method can be used to generate a NOC. Then, the NOC performance must be validated by simulating the complete SOC (i.e. NOC and IPs) because the behaviours of IPs and NOC may be interdependent and influence each other. Simulation of a single trace is relatively slow, and the number of traces is huge. Therefore, given that not all possible traces can be simulated, no 100% guarantee can be given that performance requirements are met. Moreover, the performance observed in the simulated traces and the worst-case performance of a system may differ substantially, which means that adding a "safety margin" (e.g. sizing a buffer to twice the maximum observed during simulation) is not safe (e.g. see Section 3.6).

Only analysis can cover all cases. However, the distributed arbitration in NOCs often leads to statistical performance models [1,9], which offer no guarantee that performance requirements are always met.[1] NOCs *with guaranteed services* take provisions in their architectures to offer connections with guaranteed performance, such as absence of data loss, minimum throughput, and maximum latency. This enables analytical reasoning about NOC performance. Examples are Æthereal [6, 8], Nostrum [14], aSOC [13], using time-division-multiple-access (TDMA) schemes, and [5, 12] using (virtual)-circuit-switching schemes.

NOC communication guarantees have several positive effects on the design flow. First, all IPs and the NOC are decoupled [19], meaning that the communication behaviour of one IP cannot affect that of other IPs. As a result, they can be designed and validated independently (compositionality). (In contrast to best-effort NOCs where all IPs and NOC have to simulated together.) Second, the NOC performance model can be used to generate an application-specific NOC that meets the communication requirements under all circumstances (correct by construction). Third, the performance of

[1] With 99.9% of packets meeting their required service [1], for every high-definition video frame 2000 pixels are too late. Delayed control traffic (e.g. programming a DMA engine for every 100Hz frame) can have much larger impact, and would occur every 10 seconds.

Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'05)
1530-1591/05 $ 20.00 IEEE

any given NOC (hand made, or generated by a tool) can be analytically verified to fulfill application requirements or not, instead of using simulation. As a result, verification time is shortened, and fewer design cycles are necessary. However, the use of guaranteed services relies on the explicit description of the communication requirements (or behaviour) of the IPs. This information is normally already available as part of the specification of SOC.

Overview

In this paper we describe a design flow that addresses the two problems that we identified above: the need for tools to quickly and efficiently generate application-specific NOCs, and the requirement for SOC and NOC performance validation. In Section 2 we describe the prerequisites for our NOC design flow. In Section 3 we define the design flow and explain its inputs (e.g. application requirements), outputs (NOC hardware and software, and resulting performance), and details of the individual tools (generation, configuration, verification, and simulation). We apply our design flow in Section 4, where we generate several NOCs for a MPEG SOC. For 16 IPs and a single task graph containing 21 guaranteed connections, the tools automatically dimension and generate the SystemC and VHDL for a NOC with 3 routers and 6 NIs, and 21 traffic generators and their mapping. Including the configuration, buffer sizing and performance verification, this takes less than a minute. We review related work (Section 5), and conclude in Section 6.

2 NOC Design Flow Prerequisites

In this section we first describe the prerequisites for a design flow, independent of the NOC services.

To be able to generate an application-specific NOC, the NOC must be modular, i.e. be constructed of simpler, re-usable parametrised components: the router and network interface (NI). At *design time*, these components must be instantiated and connected in an appropriate topology. Moreover, the IP ports must be connected to particular NI ports (mapping). The result is a structural description (hardware) of the NOC. The router and NI of the Æthereal NOC have been documented in [17, 18], here we mention only the relevant features. For the purposes of the design flow, the router is parametrised by its arity (number of input and output ports), and the best-effort queue sizes. Here, we fix the router link width to 32 bits. The NI is parametrised by the number of ports (to which IP ports can be connected, as specified in the mapping), the number of connections per port, and the buffer sizes per connection. The type of the IP and NI ports (AXI, various DTL profiles, their word width, etc.) is also a parameter, but kept fixed in this paper. The NOC as a whole is parametrised by the size of the slot table, and by the operating speed (500MHz in all examples, which is the speed of the router and NI implementations).

All instances of the Æthereal NOC are *(re)configurable at run time*. This means that the NIs can be (re)programmed at run-time, using standard memory-mapped IO ports on the NOC, to support a variety of connections [17]. (Routers are stateless and require no configuration.) Within the NOC's hardware limits (number of connections per port, slot table size, credit counter bit widths, etc.) connections can be configured with different (guaranteed) properties, such as throughput and latency, by programming the path from master to slave, the number of slots and flow-control credits, etc. A configuration for a use case, is a list of NOC memory registers and their values.

The Æthereal NOC offers both best-effort and guaranteed services. The design flow described in this paper can be used for any

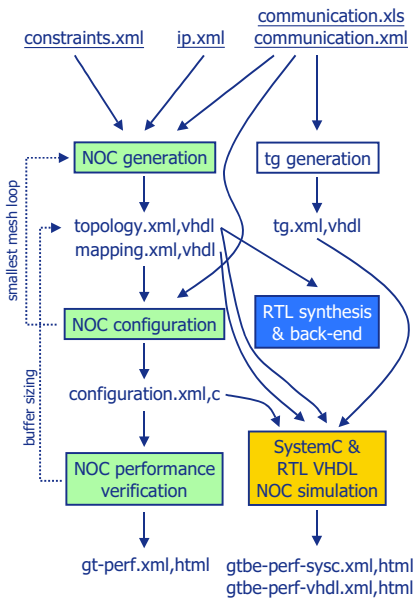


Figure 1. The Æthereal NOC Design Flow

mix of services. However, the advantage of NOCs with guaranteed services, as discussed at length in the introduction, is that the they implement router and NI arbitration schemes that allow analytical reasoning about the performance of guaranteed connections independently of the behaviour of other connections. This prerequisite is essential for correct-by-construction NOC generation and configuration, as well as compositional NOC performance verification (of any NOC, hand-made or generated), see Sections 3.3 to 3.5.

The final prerequisite for a design flow is the description of the application communication requirements. It is not possible to generate a NOC without knowing what the requirements of the application using it will be. This will be described in Section 3.1 because this information is given as an input to the design flow.

The prerequisites are therefore: a modular NOC offering guaranteed services with parametrised components (router, NI), and a description of the application requirements. The next section uses these foundations to offer a NOC design flow.

3 NOC Design Flow

Figure 1 shows the NOC design flow, which is fully implemented. Input files are underlined and shown at the top. The tools that we will discuss are shown by boxes; for simplicity some format conversion tools are not shown (in particular xls→XML, XML→VHDL, and XML→HTML). Below, we discuss each of the files and tools in turn. Although a major motivation for NOCs is their promise to improve back-end issues, such as global timing closure, we omit details of the "RTL synthesis and back-end."

First, however, note that NOC generation and configuration are interdependent, and part of one complex optimisation problem (find topology, mapping, and throughput assignments that minimise the number of routers, NIs, buffer sizes, and latencies). If this is done correctly (by construction), no performance verification and simulation is required (for guaranteed connections). Simulation is still useful, e.g. to check if the communication behaviour of IPs has been correctly characterised. With guaranteed services, this can be checked independently for every connection.

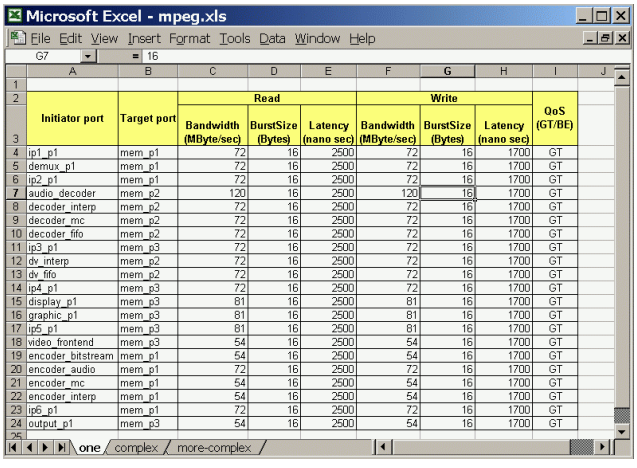
Nonetheless, our design flow has been split into separate tools (generation, configuration, verification) for several reasons. First,

breaking the design flow in smaller steps, simplifies steering or overriding heuristics used in each of the individual tools, enhancing user control. Second, it reduces the complexity of optimisation problem, and simpler, faster heuristics can be used. Higher-level optimisation loops involving multiple tools can then be easily added, such as the "smallest mesh loop," cf. Section 3.3. Third, parts of the flow can be more easily customised, added, or replaced by the user to tailor the flow or improve its performance. For example, mesh XY routing can be replaced by load-balancing turn-prohibiting routing. Finally, redundancy in the sense of checking what should be generated automatically and correct by construction, such as simulation and performance verification (of guaranteed connections), minimises impact of potential programming errors, and acts as a safety net when allowing the user to manually create or modify intermediate results.

The design flow is very simple for the user. It is based on a makefile with few targets corresponding to the major activities: generate, configure, gtverify, simsystemc, and simvhdl. All files are in XML, which is human readable, but also robust and extensible. The user can use the flow in *fully automatic mode* (supply only the required underlined input files), or *manually create or modify* selected intermediate files (topology.xml, mapping.xml, configuration.xml, tg.xml). In case of manual intervention the remainder of the flow works automatically. For example, an automatically generated NOC can be configured manually, yet still have its performance automatically verified. The input files contain all information (options, settings, etc.), for deterministic batch-mode replay of results.

3.1 Specification of the Application Requirements

The starting point of the design flow is the description of the application's communication requirements (communication.xls,xml). An application consists of a number of task graphs, or use cases. Each of these contains a number of tasks, to be executed in hardware or software, using storage, and communicating using the NOC. For the design flow only the communication is relevant, i.e. which ports on which IPs communicate with each other. Figure 2 shows an example specification in Microsoft Excel. This



Initiator port	Target port	Read			Write			QoS (GT/BE)
		Bandwidth (MByte/sec)	BurstSize (Bytes)	Latency (nano sec)	Bandwidth (MByte/sec)	BurstSize (Bytes)	Latency (nano sec)	
ip1_p1	mem_p1	72	16	2500	72	16	1700	GT
demux_p1	mem_p1	72	16	2500	72	16	1700	GT
ip2_p1	mem_p1	72	16	2500	72	16	1700	GT
audio_decoder	mem_p2	120	16	2500	120	16	1700	GT
decoder_interp	mem_p2	72	16	2500	72	16	1700	GT
decoder_mc	mem_p2	72	16	2500	72	16	1700	GT
decoder_fifo	mem_p2	72	16	2500	72	16	1700	GT
ip3_p1	mem_p3	72	16	2500	72	16	1700	GT
dv_interp	mem_p2	72	16	2500	72	16	1700	GT
dv_fifo	mem_p2	72	16	2500	72	16	1700	GT
ip4_p1	mem_p3	72	16	2500	72	16	1700	GT
display_p1	mem_p3	81	16	2500	81	16	1700	GT
graphic_p1	mem_p3	81	16	2500	81	16	1700	GT
ip5_p1	mem_p3	81	16	2500	81	16	1700	GT
video_frontend	mem_p3	54	16	2500	54	16	1700	GT
encoder_bitstream	mem_p1	54	16	2500	54	16	1700	GT
encoder_audio	mem_p1	72	16	2500	72	16	1700	GT
encoder_mc	mem_p1	54	16	2500	54	16	1700	GT
encoder_interp	mem_p1	54	16	2500	54	16	1700	GT
ip6_p1	mem_p1	72	16	2500	72	16	1700	GT
output_p1	mem_p3	54	16	2500	54	16	1700	GT

Figure 2. MPEG Application Communication Specification.

is the de facto format for design documentation, and readily available from SOC designers. The Excel document is translated to XML, which the user can also write directly. An Excel document represents a single application, and each worksheet represents a single use case. (Currently, multiple use cases are entered as independent applications.) A use case is specified as a list of connections. A connection specifies a communication between a master port and a slave port, the required (minimum) bandwidth, the (maximum) allowed latency, and burst size for read and/or write data, and the traffic class (best-effort or guaranteed).

3.2 Specification of the IP

The second input file is the specification of the architecture around the NOC. The ip.xml file, an example of which is shown in Figure 3, contains a list of all IPs connected to the NOC and the IP ports. Each port has a number of attributes, such as protocol (AXI, various DTL profiles), and data word width. (Currently, all ports are of type DTL MMIO (memory-mapped IO) or MMBD (memory-mapped block data), with 32-bit data words.) The ip.xml file is used to generate the right protocol-conversion shells for NIs [17].

```
<architecture id="MPEG">
  <IP id="display">
    <initiator id="p1"     protocol="MMBD" word="32"/>
  </IP>
  <IP id="decoder">
    <initiator id="interp" protocol="MMBD" word="32"/>
    <initiator id="mc"     protocol="MMBD" word="32"/>
    <initiator id="fifo"   protocol="MMBD" word="32"/>
  </IP>
```

Figure 3. Part of the IP description of MPEG example.

3.3 NOC Generation and IP Mapping

The first tool in the design flow is the NOC dimensioning and generation and IP mapping tool. It uses the application communication specification communication.xml, the IP specification ip.xml, and the NOC generation constraints constraints.xml. The tool defines the *design-time hardware* topology.xml: the number of routers, network interfaces, and topology. Parameters are specified for the NOC (flit duration, number of slots in TDMA table), for each router (arity, best-effort buffer size), and for each NI instance (number of NI ports, connections per port, buffer sizes per connection). To reduce NOC cost, all routers and NIs are dimensioned precisely for the application, giving many different router and NI instances per NOC. Modular router and NI architectures are therefore essential. Figure 4 shows a partial topology.xml.

```
<AENetwork id="MPEG" flitClk="6" slots="128">
  <AERouter id="R0000" iq="8">
    <AEPort id="NI"    link="L_0000" />
    <AEPort id="South" link="L_0000_0100" />
    <AEPort id="West"  link="L_0000_0001" />
  </AERouter>
  <AENI id="NI0101">
    <AEPort  id="Router"       link="L_0101" />
    <SlaveP  id="CONFIG"       conn="1" iq="4"  oq="4"/>
    <MasterP id="display.p1"   conn="1" iq="40" oq="21"/>
    <MasterP id="decoder.mc"   conn="1" iq="40" oq="21"/>
    <MasterP id="decoder.fifo" conn="1" iq="40" oq="21"/>
    ...
  </AENI>
```

Figure 4. Part of the Topology Description of MPEG example.

A synthesisable RTL VHDL description of the NOC is also produced, in a form compatible with the standard Philips back-end design flow. An area estimate of the NOC is given (using a model calibrated with existing router and NI implementations [17, 18]).

The buffers per connection in the NIs are dimensioned to avoid stalling of data by hiding the round-trip delay of credits for lossless connections, and to compensate for difference in master and

slave burst sizes. Although they are part of the hardware, they are not computed at this point in the design flow, because they depend on the configuration, which is computed later. As an example of the flexibility of the design flow, for guaranteed connections, the buffer sizes computed by the NOC performance validation tool can be back-annotated in the topology.xml file, as indicated by the dashed arrow labelled "buffer sizing" in Figure 1.

A second output is the mapping.xml file, containing the assignment of IP ports to NI ports. The mapping has a significant impact on the size (i.e. cost) of the NOC and its performance (e.g. Table 1). The constraints.xml file allows the user to influence the mapping by specifying if sets of ports must be mapped on the same NI or must be mapped on different NIs, to reflect e.g. floor-planning constraints. For example, `<SameNI> <Module id="display"> <Module id="decoder" port="mc"/> </SameNI>` places all IP ports of the display IP, and the decoder's mc port of Figure 4 on the same NI.

The NOC topology can be computed in three modes: either a mesh of given size is generated with a IP mapping, or the smallest mesh and IP mapping accommodating the application are generated, or a user-defined topology and IP mapping can be used. Automatic shortest-path routing can be used in all cases, and XY routing also in the first two cases. The computation of the smallest mesh and IP mapping for the given application, depends, like the buffer computation, on the configuration (e.g. the heuristic slot allocation may fail). The design flow therefore implements the automatic loop (the dashed arrow labelled "smallest mesh loop" in Figure 1) as the makefile target "minmesh." Determining the maximum slot table size, which is a hardware constant but also depends on the configuration, can be computed with a similar loop.

Briefly, the mapping algorithm works as follows, for both best-effort and guaranteed connections. It balances the IP port bandwidths over the NIs, clustering IP ports that communicate heavily on the same NI, and then minimising the distance (number of hops) between heavily communicating NIs, taking care not to overload any link. Packetisation overhead and latency constraints are ignored at this point.

3.4 NOC Configuration

The generation and mapping tool produces the design-time hardware (topology.xml and mapping.xml). Using these, the NOC configuration tool computes the *run-time software* that contains all the information to program the hardware. The configuration.xml file contains the values for all programmable registers of the NIs (the routers are stateless), such as connection identifiers, and for each connection, the path from master to slave port, and flow control credits. For connections with guaranteed throughput or latency (as specified by the user in communication.xls), a slot allocation must be determined (i.e. each hop along the path the slot increases by one, and at most one connection can use a given slot at a router [18]).

The configuration algorithm works as follows. For each connection a path is generated, like by the mapping tool (or it can be supplied by the user). The flow control credits are equal to buffer sizes. Slots are allocated using a heuristic using a combination of each connection's path length and required bandwidth.

Figure 5 shows a partial configuration in (equivalent) XML and C. Section 3.6 describes how the configuration.xml and configuration.c files are used by the SystemC and RTL VHDL simulations.

```
<Connection master="decoder.mc" cidm="0"
            slave="mem.p2" cids="2">
  <Request type="GT" path="3 1 0" credits="33"
           slots="22 23 24 25 26 27 28 29 30 31 32"/>
  <Response type="GT" slots="7 8 9 10 11 12 13"
            path="2 1 0" credits="21"/>
</Connection>
```

```
open_connection ("decoder.mc",0,"mem.p2",2,
                 "GT","22-32","3 1 0",33,
                 "GT","7-13","1",60);
```

Figure 5. Part of the MPEG example Configuration (XML and C).

Using the topology and configuration the worst-case and average energy consumption of the NOC can be estimated [4]. This, and the area estimate of the NOC, computed from the topology, are important indicators of NOC *cost*. The verification and simulation steps, discussed below, compute the NOC *performance*.

3.5 NOC Verification

The rationale for decomposing the design flow into smaller tools has been given at the start of this section. Thus, the NOC hardware (topology.xml) and software (configuration.xml) are computed sequentially, earlier in the design flow. Therefore, the generation, mapping, and configuration tools must work with some simplifying assumptions (e.g. that NI buffers are adequately sized, ignore latency constraints). As a result, configuration (for guaranteed connections) may fail for a NOC topology and mapping, and latency constraints may not be met for a configured NOC.

For best-effort connections, the latter can only verified by simulation (next section). For guaranteed services, on the other hand, Æthereal's TDMA performance model can be analysed mathematically, and throughput, latency, and buffer sizes can be computed for the worst case [6]. Therefore, although the generation and configuration tools do not generate NOCs that are correct by construction for the specified application, the final verification step checks whether the NOC topology and configuration are guaranteed to fulfill the application requirements or not. *Any Æthereal topology and configuration, automatically or manually generated, can be verified against its application requirements.*

Given the topology.xml, mapping.xml, configuration.xml files the verification tool computes the worst-case (minimum) throughput, (maximum) latency, and (minimum) buffer sizes per connection. These are compared to the requirements (communication.xml) and shown in an intuitive colour-coded table (gt-perf.xml,html), cross-linked with communication.xml,html (Figure 6). Green entries meet the requirements, red ones do not. Yellow buffer sizes meet the requirements but are overdimensioned. It is possible (and, in fact, the norm) to automatically back-annotate the computed buffer sizes in the topology.xml file, as can be seen from the zero slack in the figure.

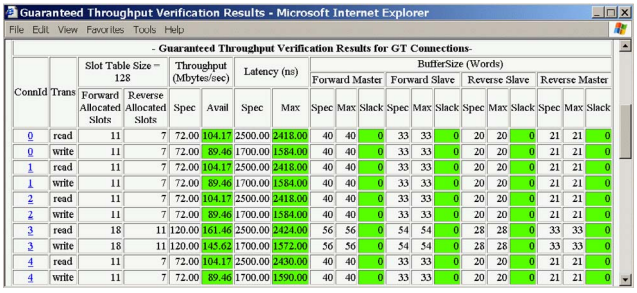


Figure 6. Performance Verification Output of MPEG example.

The verification checks each GT connection independently.

This compositionality is a useful consequence of Æthereal's TDMA model. The model is also relatively simple to reason about [6]. As an example, a lossless (flow-controlled) request-only connection (e.g. writes without acknowledgements) consists of a outgoing (master→slave) channel for requests and a return (slave→master) channel for flow control. First, compute the amount of write data that can be sent in one slot table revolution. Then compute the worst-case duration to send a complete burst (this may take several slot table revolutions, and includes the worst-case wait for the initial slot). Add to this the latency in the routers (flit delay times the length of connection's path). As a complicating factor, the availability of request credits must be taken into account: this depends on the bandwidth and latency of the return channel, computed similarly. The sizes of the buffers at the master and slave for both outgoing and return channels are derived from the above computation. Note that to guarantee throughput or latency for a connection with only request transactions, the bandwidth and buffer sizes and credits must be sufficient on both the outgoing and return channels. Other types of connections are verified similarly.

3.6 NOC Simulation

The verification tool works for guaranteed connections only, and not for best-effort connections. Moreover, it computes worst-case throughput and latency numbers and buffer sizes, not actual or average figures. To assess the average performance of both guaranteed and best-effort connections for a particular execution trace two types of simulation are supported: RTL VHDL and SystemC. The former is a bit and cycle-accurate simulation of the RTL VHDL implementation. In the SystemC simulation, the NOC is simulated at the flit level, and the IP-NI interface is at the transaction level. The SystemC simulation is orders of magnitude faster than the VHDL simulation.

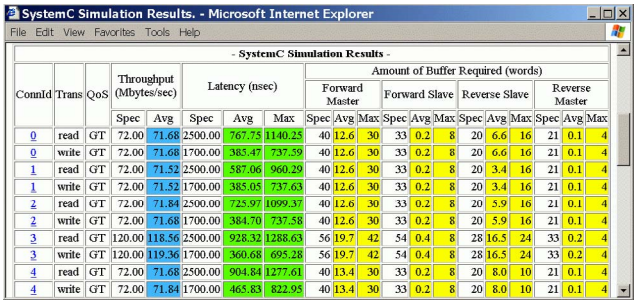
Simulation must be flexible to cope with automatically generated and configured NOCs. For this reason, our SystemC simulation is based on XML files for topology, mapping, NOC configuration, and IP configuration, *to enable fast run-time* NOC *and* IP *instantiation and configuration* [7].

The configuration is produced as an XML and C file (Figure 5). The XML file is used by the SystemC and VHDL simulation. The Æthereal NOC is configured through memory-mapped IO (MMIO) NOC ports (the `id="CONFIG"` port in Figure 4), using the NOC itself [17]. In the VHDL simulation a behavioural IP configures the NOC, based on configuration.xml. The SystemC simulation can use configuration.xml directly, or model the MMIO programming.

The configuration.c file is intended to be run on an embedded processor connected to the NOC. The C file contains a function per use case, e.g. `configure_usecase_one()`, that (re)programs the NOC at run-time, using the MMIO port of processor's NI. For every connection an Æthereal library function (`re/open/close_connection`) is called with the appropriate parameters (e.g. path, slots, credits; see Figure 5).

Traffic Generation and Measurements

SOC simulation requires that all IPs are available for simulation, as a model (e.g. at a behavioural or cycle-accurate level) or as an implementation. Early in the design cycle, this is usually not the case. To enable early SOC simulation to evaluate the NOC performance, our design flow produces XML-configurable SystemC and VHDL *traffic generators*, see Figure 1. All IP ports of IPs for which no model or implementation is available, are modelled with traffic generators. The traffic is generated according to the IP requirements as given in the communication.xls file. Traffic generators can be replaced by IP models, as these become available. At all times, however, a complete SOC can be simulated, in both SystemC and VHDL.


SystemC Simulation Results. - Microsoft Internet Explorer

File Edit View Favorites Tools Help

- SystemC Simulation Results -

ConnId	Trans	QoS	Throughput (Mbytes/sec)		Latency (nsec)			Amount of Buffer Required (words)											
								Forward Master			Forward Slave			Reverse Slave			Reverse Master		
			Spec	Avg	Spec	Avg	Max	Spec	Avg	Max	Spec	Avg	Max	Spec	Avg	Max	Spec	Avg	Max
0	read	GT	72.00	71.68	2500.00	767.75	1140.25	40	12.6	30	33	0.2	8	20	6.6	16	21	0.1	4
0	write	GT	72.00	71.68	1700.00	385.47	737.59	40	12.6	30	33	0.2	8	20	6.6	16	21	0.1	4
1	read	GT	72.00	71.52	2500.00	587.06	960.29	40	12.6	30	33	0.2	8	20	3.4	16	21	0.1	4
1	write	GT	72.00	71.52	1700.00	385.05	737.63	40	12.6	30	33	0.2	8	20	3.4	16	21	0.1	4
2	read	GT	72.00	71.84	2500.00	725.97	1099.37	40	12.6	30	33	0.2	8	20	5.9	16	21	0.1	4
2	write	GT	72.00	71.68	1700.00	384.70	737.58	40	12.6	30	33	0.2	8	20	5.9	16	21	0.1	4
3	read	GT	120.00	118.56	2500.00	928.32	1288.63	56	19.7	42	54	0.4	8	28	16.5	24	33	0.2	4
3	write	GT	120.00	119.36	1700.00	360.68	695.28	56	19.7	42	54	0.4	8	28	16.5	24	33	0.2	4
4	read	GT	72.00	71.68	2500.00	904.84	1277.61	40	13.4	30	33	0.2	8	20	8.0	10	21	0.1	4
4	write	GT	72.00	71.84	1700.00	465.83	822.95	40	13.4	30	33	0.2	8	20	8.0	10	21	0.1	4

Figure 7. SystemC Simulation Output of MPEG example.

The traffic generators also automate the measurement of throughput and latency of all connections in the NOC, for both SystemC and VHDL, and NI buffer statistics (SystemC only). The SystemC and/or VHDL results are reported in colour-coded tables (gtbe-*-perf.xml,html), cross-linked with communication.xml,html (Figure 7). Like the verification output (Section 3.5) green entries meet the requirements, and red ones do not. Due to simulation artifacts (random number generators in traffic generators, run-in effects due to insufficiently long simulation, etc.) a guaranteed connection may not use its required bandwidth (e.g. write data of connection 0). Note that this is due to the traffic generators, not the NOC: the bandwidth and latency of guaranteed connections is available, but it is not used.

Note that using simulation as a basis for e.g. buffer dimensioning can lead to buffers that are too small. Analytical performance evaluation shows that connection 3's write data requires a buffer of 54 words, but the maximum filling during simulation is only 8. Overdimensioning all buffers by the same factor of $7 = \lceil 54/8 \rceil$, would lead to buffers that are up to 6 times too large (7x24 instead of 28, for connection 3), doubling the NOC size ("simulation," in Table 1). Of course, without verification, the factor 7 would have to be guessed in the first place.

4 MPEG Codec Case Study

We applied the design flow to an existing MPEG codec SOC with 16 IPs. The architecture uses a single external SDRAM with three ports to implement all communication between IPs. We specified the read and write bandwidths and latencies of the 21 guaranteed-throughput connections (Figure 2). Running the design flow in *fully automatic mode* (generating the minimum mesh and smallest slot table, with back-annotated minimum buffer sizes) resulted in a 2x3 mesh with 128 slots, with a total estimated area of 2.35 mm^2 in a 0.13 micron process, in minutes (cf. Table 1). Figures 4-7 refer to this NOC instance. For comparison, a naive mapping (one IP per NI) is almost double the size. "Simulation" shows that using simulation and a "safe" overdimensioning factor to determine buffer sizes (see Section 3.6), is more than twice as big. The average worst-case write latency per connection, computed by the performance verification tool, is more or less constant.

To highlight the scope for *manual intervention*, we also optimised the NOC. We explored several different IP-NI mappings to improve the IP distribution and clustering, to reduce the constraints on the routing and slot table configuration. A 21% smaller

Table 1. Comparison of MPEG NOCs.

generation	mesh	slots	NI area	router area	total area	area diff	avg wc latency
automatic	2x3	128	1.83	0.51	2.35	ref	1570 ns
naive	3x6	128	2.17	2.32	4.49	+91%	1583 ns
simulation	2x3	128	4.61	0.51	5.13	+118%	1570 ns
optimised	3x1	8	1.51	0.35	1.86	-21%	399 ns

mesh (3x1, 1.86 mm^2) then suffices. Despite the resulting bandwidth overallocation, it also reduces the slot table (8 slots), and the latency (almost four-fold reduction of average worst-case write latency) and buffer sizes (compare Figures 6 and 8). The minimum buffer sizes were computed by the verification tool and automatically back-annotated in the custom topology.

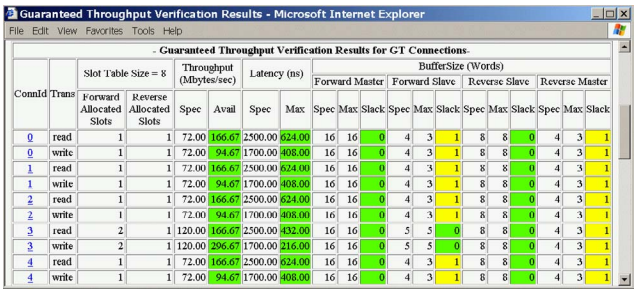


Figure 8. Performance Verification of Optimised MPEG.

The case study shows that the NOC design flow work generates a RTL VHDL implementation of a NOC, which has a guaranteed performance for the MPEG codec SOC. Retrofitting a NOC to an existing SOC architecture is possible, although for example, the single external memory underutilises the parallelism offered by NOCs. It also shows that manual optimisation gives worthwhile improvements for a small SOC. We are currently evaluating the design flow for a very large SOC, and will see if manual optimisations are still feasible (many IPs and many use cases) and significant.

5 Related Work

QNoC's design flow [2] is similar to ours, but differs in relying on system simulation to verify SOC performance, expressed as statistical properties. They first generate the NOC topology generation and NI-IP mapping, followed by balancing bandwidth in the NOC, whereas we perform both at the same time.

XpipesCompiler [11] and NcGEN [3] generate optimised SystemC (and VHDL for NcGEN) descriptions for manually specified NOC topologies, which can be simulated together with traffic generators for performance validation.

[10, 15, 16] synthesise NOCs, map IPs, and/or configure NOCs based on bandwidth requirements and link bandwidths. However, when using worm-hole routing, the capacity of the links is not independent, especially in the presence of links of different speeds. (e.g. a packet spanning a slow and a fast link will reduce the capacity of the fast link, in the absence of virtual-channel buffering.) Applying these sophisticated synthesis and mapping algorithms to NOCs with guaranteed services, requiring the inclusion of NOC configuration, will be very beneficial for design flows, and speed up SOC and NOC design.

6 Conclusions

We have described a *flexible operational design flow to generate application-specific network-on-chip (*NOC*) instances and configurations*. The application's requirements are specified in Excel, which is readily available from system architects. Application-specific NOC instances, consisting of routers and network interfaces (NIs) are automatically dimensioned and generated. An IP-NI port mapping is also computed. The result is SystemC and synthesisable RTL VHDL, compliant with the Philips back-end flow.

The NOC hardware is *run-time (re)programmable* to support different task graphs. The configuration (software) to program the network is generated in XML (for SystemC and VHDL simulations) and in C for embedded processors that program the NOC using memory-mapped IO. The NOC hardware and configuration can be simulated in SystemC and VHDL, using custom traffic generators. These mimic the IP behaviour, as specified in the application requirements; thus, the SOC can be simulated at all times.

A unique feature of our design flow is the *fast automatic performance verification*: given the NOC hardware and configuration, the guaranteed minimum throughput, maximum latency, and minimum buffer sizes are analytically computed for all guaranteed connections. The guaranteed communication services of Æthereal are essential to achieve this. Any NOC instance and configuration can be verified, whether automatically or manually created. Analytical performance verification eliminates lengthy simulations for the guaranteed connections, and hence reduces verification time, leading to fewer design cycles.

Using the design flow we automatically generated and verified a NOC for an MPEG codec SOC (2x3 mesh, 2.35 mm^2 in 0.13 micron) in minutes. The design flow's flexibility was demonstrated by manually optimising a NOC, leading to a 21% area reduction.

Acknowledgement. S. González Pestana is supported by Marie Curie Fellowship IST-GH-99-80002-02.

References

[1] E. Bolotin et al. QNoC: QoS architecture and design process for network on chip. *J. of Systems Architecture*, 50(2–3):105–128, 2004.
[2] E. Bolotin et al. Automatic hardware-efficient SoC integration by QoS network on chip. In *ICECS*, 2004.
[3] J. Chan et al. NoCGEN: a template based reuse methodology for networks on chip architecture. In *Proc. Int'l Conference on VLSI Design*, 2004.
[4] J. Dielissen et al. Power measurements and analysis of a network on chip. Submitted, 2004.
[5] T. Felicijan et al. An asynchronous on-chip network router with quality-of-service (QoS) support. In *SoCC*, 2004.
[6] O. P. Gangwal et al. Building predictable systems on chip: An analysis of guaranteed communication in the Æthereal network on chip. In P. van der Stok, editor, *Dynamic and Robust Streaming In and Between Connected Cnsumer-Electronics Devices*. Kluwer, 2005.
[7] S. González Pestana, et al. Cost-performance trade-offs in networks on chip: A simulation-based approach. In *DATE*, 2004.
[8] K. Goossens et al. Guaranteeing the quality of services in networks on chip. In A. Jantsch and H. Tenhunen, editors, *Networks on Chip*. Kluwer, 2003.
[9] P. Guerrier. *Un Réseau D'Interconnexion pour Systémes Intégrés*. PhD thesis, Université Paris VI, Mar. 2000.
[10] J. Hu et al. Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints. In *DATE*, 2004.
[11] A. Jalabert et al. XpipesCompiler: A tool for instantiating application specific networks on chip. In *DATE*, 2004.
[12] N. Kavaldjiev et al. A virtual channel router for on-chip networks. *SoCC*'04.
[13] J. Liang et al. aSOC: A scalable, single-chip communications architecture. In *PACT*, 2000.
[14] M. Millberg et al. Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip. In *DATE*, 2004.
[15] S. Murali et al. SUNMAP: A tool for automatic topology selection and generation for NOCs. In *DAC*, 2003.
[16] A. Pinto et al. Efficient synthesis of networks on chip. In *ICCD*, 2003.
[17] A. Rădulescu et al. An efficient on-chip network interface offering guaranteed services, shared-memory abstraction, and flexible network programming. In *DATE*, 2004.
[18] E. Rijpkema et al. Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip. In *DATE*, 2003.
[19] M. Sgroi et al. Addressing the system-on-a-chip interconnect woes through communication-based design. In *DAC*, 2001.