



## Benchmarking mesh and hierarchical bus networks in system-on-chip context

Erno Salminen <sup>a,\*</sup>, Tero Kangas <sup>b</sup>, Vesa Lahtinen <sup>b</sup>, Jouni Riihimäki <sup>b</sup>,  
Kimmo Kuusilinna <sup>c</sup>, Timo D. Hämäläinen <sup>a</sup>

<sup>a</sup> *Tampere University of Technology, Institute of Digital and Computer Systems, P.O. Box 553, FIN-33101 Tampere, Finland*

<sup>b</sup> *Nokia Technology Platforms, Tampere, Finland*

<sup>c</sup> *Nokia Research Center, Tampere, Finland*

Received 31 March 2006; received in revised form 10 November 2006; accepted 10 November 2006

### Abstract

The performance and area of a System-on-Chip depend on the utilized communication method. This paper presents simulation-based comparison of generic, synthesizable single bus, hierarchical bus, and 2-dimensional mesh on-chip networks. Performance of the network depends heavily on the application and therefore six test cases with multiple parameter values are used. Furthermore, two versions of each network topology are compared. The results show that hierarchical bus scales well to large number of agents and offers a good performance and area trade-off although it has smaller aggregate bandwidth and area than mesh. Hierarchical HIBI bus achieves runtimes comparable to 2-dimensional cut-through mesh with about 50% smaller network logic. However, depending on the test case, the runtime can be reduced by 20–50% when wider bus links are utilized.

© 2006 Published by Elsevier B.V.

**Keywords:** On-chip network; Simulation; Benchmark; Hierarchical bus; Mesh

### 1. Introduction

The increasing complexity of digital systems has forced the adoption of modular design techniques and the re-use of pre-designed and pre-verified components in the design process of System-on-Chips (SoC). Due to increasing number of connected components, the communication and interconnect wiring are becoming a serious problem [1–3]. Several

architecture and circuit-level alternatives have been proposed to solve this but proposals frequently only deal with the theoretical limitations of the communication networks. However, the practical limitations and requirements for the networks are affected by system-level issues and the data transfer distributions of the targeted applications.

The optimal on-chip network (also dubbed network-on-chip, NoC) has been subject to debate over the last few years. Traditionally, such networks are based on circuit-switched techniques using various topologies, such as point-to-point, bus-based networks [4], crossbars [5], and 2-dimensional meshes

\* Corresponding author.

E-mail addresses: [erno.salminen@tut.fi](mailto:erno.salminen@tut.fi) (E. Salminen), [timo.d.hamalainen@tut.fi](mailto:timo.d.hamalainen@tut.fi) (T.D. Hämäläinen).

[6,7]. Packet-switched networks have also been proposed with many topologies, such as fat-trees [8], 2-dimensional meshes [9–11], and rings [12]. An excellent textbook on the networks in general is [13]. Table 1 lists some comparisons of different on-chip networks found in literature. All the listed utilize simulation, except [14] that uses mathematical models ([9] uses both). For each survey, table shows which topologies are compared. The topologies that are evaluated only in one reference are grouped under category “other”. Note that the number of networks includes also the different versions of the same topology. Furthermore, the number of evaluated system sizes is shown. Analytical models are denoted with  $f()$ . Many studies concentrate on only one system size (i.e. the number of processing elements is fixed) and vary other parameters (e.g. buffer sizes). Many comparisons examine single bus or multiple parallel buses [17] but omit the more versatile hierarchical bus structures.

Networks are sometimes compared by the theoretical maximum transfer capabilities. This is straightforward but the results seldom reflect the performance of real applications accurately, i.e. the order of performance values may be defined but not their ratio. A single test case is not enough to have reliable performance estimates, but a set of benchmarks is needed. Applications give the best accuracy but are harder to port to different systems. Liang et al. use real applications, such as MPEG-2 encoder and Doppler radar signal analysis [6], whereas Zhang et al. use small computation kernels, such as dot product, vector sum, and IIR [17]. Pimentel et al. use motion JPEG [21]. All other surveys rely on traffic generators. Building actual applications is laborious only for comparison purposes and their simulation is time-consuming. Furthermore, the runtime of an application is often dependent on input data. Note that the test case counts in the table are merely suggestive since it is not clear when the change in some parameter actually defines a new test case.

Traffic generators provide flexibility and accelerate simulation with the cost of reduced accuracy. They use transfer patterns, or profiles, that resemble the external behavior of real applications [18]. In statistical generators [15,19,20], the communication profile is often independent of the profiles of other agents (processing elements) that are connected to the network. The transfer probability is a function of all application tasks mapped onto

Table 1  
Some comparisons of on-chip networks in literature

First author	Ref.	Topologies							Test cases					Criteria						
		Mesh	Bus	Hier. bus	Fat-tree	Ring	Other	# networks	# system sizes	Statistical	Transfer-dep.	Application	Runtime	Area	Latency	Throughput	Power/energy	Other	# criteria	
Liang	[6]	x	x	x				3	5			5	x		x					2
Wiklund	[7]	x						1	1		2				x			x		2
Andriahantainaina	[8]		x		x			2	4		1									2
Bolotin	[9]	x	x				x	4	$f()$		3							x		3
Moraes	[11]	x						1	2		3									3
Saastamoinen	[12]				Tree	x		2	1		1									3
Zeferino	[14]	x						2	$f()$		1									3
Lahiri	[15]	x	x	x				3	1		3									1
Kreutz	[16]	x	x		Tree			4	2											1
Zhang	[17]	x						3	49											1
Thid	[19]	x	x					2	1		1									3
T. Salminen	[20]	x						1	1		4									2
Pimentel	[21]	x						3	1		1									2
Pande	[23]	x	x		x			6	1		3									5
E. Salminen	This	x	x	x				6	4		1									4
Total	-	11	9	4	4	2	4	-	-	-	-	9	6	5	4	4	9	9	-	-

given agent. Therefore, statistical methods are suitable for analyzing and optimizing the network when the system architecture and application mapping are fixed. Transfer-dependent methods [16,21,22], in contrast, enable architecture exploration including component allocation, application mapping, and communication scheduling. Transfer dependence refers to a situation in which an agent cannot proceed before it has received certain data from other agent(s). A scalable synthetic benchmark can obtain a superset of the information given by any particular fixed size benchmark and, hence, remains valid for longer time. However, care must be taken to ensure that the artificial cases represent properties of real applications. Five studies, including this one, consider transfer dependencies. The cases may be synthetic or derived from real applications.

Most listed comparisons neglect the cost of the network in terms of area. Area is considered analytically in [9,14]. Different versions of single topology are compared with synthesis in [11,12]. The area of routers varies greatly; between 4 and 700 kilogates [11,10], and therefore it should be included in comparison. Recently, Pande et al. have presented a thorough study regarding several topologies and cost functions (including area) but with statistical traffic models only [23].

A single bus is a viable solution for systems with limited number of agents and bandwidth requirements due to its simplicity and the numerous legacy implementations. In [16], a single bus outperforms (both circuit and packet-switched) mesh and tree topologies, and the performance difference is rather large, being  $3\times$ – $7\times$ . However, contradictory to that, a single bus is found inadequate when compared to mesh [6,19,14] and to fat-tree [8,16]. It should be noted that increased network parallelism does not translate directly to increased performance. For example, a  $5 \times 5$  crossbar enables five parallel transfers and gives 50% increase in application performance compared to single bus [21]. Above studies did not consider hierarchical buses. A bus bridge may offer a  $1.1\times$ – $4\times$  speedup and similar performance as a ring [15]. On the other hand, a bus bridge may sometimes even increase the runtime as in [6] (10% on average).

This work utilizes synthetic, transfer-dependent test cases for benchmarking single bus, hierarchical bus, and 2-D mesh on-chip networks. Two versions of each topology are compared. Furthermore, the area of the network logic is examined with synthesis.

Currently there are no commonly accepted benchmark sets for on-chip networks, but the work presented in this paper could be proposed as part of such set. The utilized networks are presented in Section 2 and the transfer-dependent traffic patterns in Section 3. Section 4 examines the area costs of the implementations based on synthesis results and describes the runtime comparison. In Section 5, the conclusions of the work are given.

## 2. The evaluated topologies

To allow fair comparison, the networks are compared by using synthesizable building blocks. All networks utilize the same interface to processing element (PE). At first, all topologies were implemented using packet-switching with store-and-forward buffering. It is easy to implement but may be suboptimal regarding the latency and the required buffering area. Hence, other versions of each topology were developed. For mesh, a cut-through switching was implemented. Although wormhole switching would allow smaller buffers, it was omitted here for brevity. For buses, HIBI v.2 [24] was also used. It is a bus-based network allowing hierarchical topologies but it does not use packet-based transfers. Thus, six networks in total are compared. All networks utilize deterministic routing and hence, the data always arrive in-order.

Single shared bus architecture is usually suitable only to systems with less than 10–20 agents due to the limited bandwidth and the problems induced by the large loads and long signal lines. The traditional single bus topology can be extended to a hierarchical bus scheme as shown in Fig. 1a. Extension is done by using bridges (marked with  $b_j$ ) to connect several bus segments together. Processing elements ( $pe_i$ ) are connected to bus segments via wrappers ( $w_i$ ) and together they form an agent. In this case, each segment operates with the same frequency and has the same number of signal lines. In this paper, hierarchical bus is built as a chain of bus segments, although a tree like structure could also be used. The number of segments is scaled with the number agents. This is a distinct feature compared to many previous proposals that have a constant number of bus segments, e.g. only two or three. The implemented bus wrapper is a fairly simple device with two FIFO buffers and a control unit for arbitration. Arbitration is based on a distributed round-robin scheme. In packet-switched bus, the ownership is passed to the next wrapper after each

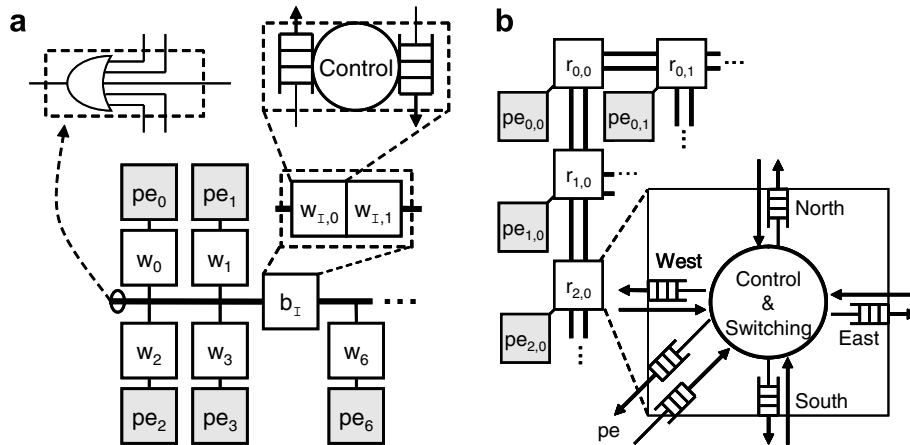


Fig. 1. Network implementations. (a) Hierarchical bus and (b) 2-dimensional 4-way mesh.

transmitted packet. In HIBI, the ownership changes when a pre-defined number of words have been transferred (40 words in this case) or when the sender runs out of data. In both bus versions, the bridge components are implemented by connecting two wrappers together and the bus signal resolution is implemented with an OR-based structure as shown in Fig. 1a. In hierarchical bus, the problems inherent in long signal wires are solved by grouping only a limited number of agents, in this case four, in each bus segment. Therefore, single bus and hierarchical bus topologies are, in this case, the same for a system with four agents.

Fig. 1b depicts a packet-switched 2-dimensional 4-way mesh used in this study. The processing elements are connected to an array where the router elements ( $r_{y,x}$ ) handle the storing and forwarding of the data. A router comprises of a control unit taking care of switching and routing, and a FIFO buffer for each direction (North, East, South, West, and two for PE). Note that West FIFO is actually removed from the highlighted router that resides on the mesh boundary. Removing such unnecessary FIFOs clearly reduces the mesh area: from 9% (64 routers) up to 49% (4 routers). Routers implement a simple dimension-order routing scheme where the transfers are first directed to the correct row and then to the requested column.

### 2.1. Static topology analysis

The basic properties of the networks are listed in Table 2. A single bus has only one communication

link, whereas a hierarchical bus has as many links as there are bus segments (each having four agents). In mesh, there are four unidirectional links from each router except on those residing on edges. Consequently, mesh has by far the highest aggregate bandwidth, i.e. the sum of the link bandwidths. In addition, rough estimates (as a function of the number of agents,  $N$ ) for the total wire length and the length of the longest wire are shown. The length is shown as multiples of the side length of the PE ( $D_{PE}$ ). Estimates assume uniformly sized, square-shaped PEs. In reality, PEs may have varying size and non-square shape and, therefore, full placement and routing would be needed to determine wire properties accurately. Furthermore, the area required by network logic should be added to PE area for determining the wire lengths.

Since the number of segments scales with the number of agents, the length of the longest wire remains constant in hierarchical bus topology. However, the total wire length is equal to  $N \times D_{PE}$  as in a single bus topology. The mesh has the shortest constant-length wires but requires 2–3.5 times more wiring area than buses for 4 and 64 agent systems, respectively. Similar analysis on wire lengths is presented in [9]. However, [9] varies the data width to achieve equal aggregate bandwidth and the number of segments is defined in a different manner. The total wire length refers to 1-bit data; the values should be simply multiplied by the desired data width. As mentioned, bus wrappers have two buffers (tx and rx), bus bridges have four buffers and mesh routers have six buffers (except those on the network boundary).

Table 2

The basic properties of the studied networks.  $N$  denotes the number of agents and  $D_{PE}$  means the side length of a processing element

Network	Number of links ( $L$ )					Total wire length, [ $D_{PE}$ ]	Longest wire, [ $D_{PE}$ ]	Number of buffers
	4	16	36	64	$N$			
Single bus	1	1	1	1	1	$N$	$N$	$2N$
Hier. bus	1	4	9	16	$N/4$	$N$	4	$3N-4$
Mesh	8	48	120	224	$4 \times (N - N^{0.5})$	$4 \times (N - N^{0.5})$	1	$6N - 4N^{0.5}$

### 3. Test setup

The comparison is done with synthetic benchmarks that are generated to represent characteristic application properties, such as sequential/parallel behavior, communication/computation intensiveness, and spatial traffic distribution. The test cases are executed in a simulation environment called Transaction Generator (TG) [22] that is independent of the network and runs application descriptions based on the Kahn process network model [25]. Each process can be either waiting for data, reading data, processing, writing data, or finished. TG notably accelerates (up to 40 $\times$ ) the simulation compared to HW/SW co-simulation with multiple instruction set simulators. At the same time, the timing error is less than 10% with respect to real application. Since the actual sizes of PEs and wire length are unknown, it is impossible to determine the maximum operating frequencies for networks accurately. For simplicity, all networks have the same operating frequency in the simulations, meaning that the runtimes are measured in clock cycles. Considering also the operating frequency would affect the single bus the most, as its maximum wire length increases with the system size (Table 2).

#### 3.1. Test cases

Fig. 2 shows the used process network graphs for eight agents. The white nodes depict computation

processes that can have an arbitrary processing time of  $P$  clock cycles. The edges represent data transmissions with length of  $D$  words. Computation at a node cannot start until at least one of the arriving transfers has completed (transfer dependence). The start processes, marked with black nodes, have  $P = 1$  and  $D = 1$  and are executed only once to trigger computation processes. By changing  $P$  and  $D$ , the application model in TG can be made more computation or communication intensive. Both  $P$  and  $D$  can be varied randomly within a user-defined range. All these benchmarks are scaled with  $N$  so that there is one computation process and variable number of start processes per agent. The dashed lines describe a simple 1-to-1 mapping of the process graph onto eight agents. However, the process graphs are totally independent of the hardware architecture and other mappings can be easily explored.

The first benchmark, case 1, (Fig. 2a) resembles a sequential data flow application having only one start process. Case 2 (Fig. 2b) is partly sequential and partly parallel in nature having  $N/2$  start processes. Case 3 (Fig. 2c) presents an application where the transfers are sequential as in case 1 but within groups of four processes so the communication is localized. Case 4 (Fig. 2d) has processes in a group of four transmitting data in parallel to each other. Cases 2 and 4 can also be thought as pipelined versions of cases 1 and 3, respectively. In addition, case 5 combines the cases 1–4 into one

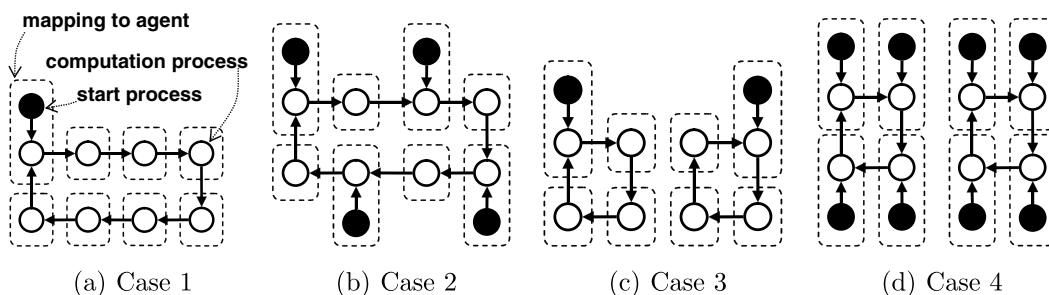


Fig. 2. Test case types used in comparison,  $N = 8$ .

Table 3

The number of start processes in test cases as they are scaled with the number of agents ( $N$ )

Case	Number of start processed ( $S$ )				
	4	16	36	64	$N$
1	1	1	1	1	1
2	2	8	18	32	$N/2$
3	1	4	9	16	$N/4$
4	4	16	36	64	$N$
5	8	29	64	113	$(7N/4) + 1$

simulation to represent heterogeneous behavior. In case 5, cases 1–4 are run together so that each agent executes one computation process from each case. For example, all cases have start and computation processes grouped together in the top left corner (cf. Fig. 2a–d) and they all are mapped to first agent in case 5. The next computation process to the right is mapped to agent 2 and so on. The mutual order of the subcases is not specified. The number of start processes is shown in Table 3.

### 3.2. Static runtime analysis

The total runtime of an application is a sum of computation time and communication time. It can be estimated in a heuristic fashion for these graphs as

$$t_{\text{tot}} = \frac{\sum P_i}{\min(N, S)} + \frac{\sum (D_i \times k)}{\min(N, L, S)}, \quad [\text{clock cycles}] \quad (1)$$

where

$$k = \frac{\text{payload} + \text{header} + \text{arbitration}}{\text{payload}} \quad (2)$$

is an implementation specific factor that is explained later. The dividers in (1) define the achievable parallelism. The parallelism of an application is defined here as the maximum number of parallel transfers and active computation processes, and it equals the number of start processes ( $S$ ) in these cases. If there are less agents ( $N$ ) or communication links ( $L$ ) than start processes ( $S$ ), the maximum parallelism of the application cannot be achieved. Similarly, the maximum number of initiated transfers per clock cycle cannot exceed the number of agents in any network. For example, case 1 is a sequential application having only one start process and utilizes only one processor or communication link at a time. Adding more communication links should

not speed up the computation at all. However, the arbitration may get simpler (smaller  $k$ ) and can thereby reduce the communication time.

The factor  $k$ , caused by arbitration and the overhead from packet headers, is calculated with Eq. (2). It is defined as the number of clock cycles needed to transfer one packet divided by the amount of transferred payload data words. Ideally  $k$  would be one. Packet and header sizes are expressed as multiples of the word size, because one word can be transferred in one clock cycle. With distributed arbitration, each agent in a bus segment has to wait whole round-robin iteration between consecutive packets. However, there can be  $S$  agents active in each round, which reduces the overall arbitration delay. Currently, the routing algorithm of a mesh simply checks one input each clock cycle for new transfers, thus, on average  $5/2$  clock cycles are needed for routing. The term *arbitration* is assumed to be  $(N - 1)/S$  for bus, 6 for hierarchical bus (4 agents and 2 bridges per segment), and 2.5 for mesh. Estimate for case 5 is a sum of individual estimates for cases 1–4. Still, (1) does not take transfer dependencies into account and assumes a uniform, localized mapping.

## 4. Results

The communication networks were implemented using RTL VHDL and synthesized using a 180 nm CMOS technology. The packet payload size was set to eight 32-bit words. Packets have a three-word header and eight-word payload data and hence the required buffer size is  $(3 + 8) \times 32$  bits in all the internal network buffers. The header includes target router ID, data amount, and address fields. The header could perhaps be compressed by concatenating target ID and amount together, but was not done here for simplicity. In contrast, HIBI requires a one-word header for address and does not utilize packets.

### 4.1. Synthesis results

The resulting network logic areas in kilogates are depicted in Table 4. The wiring area is not taken into account. The difference is mainly due to buffers (cf. Table 2). HIBI has 5-word buffers but more complex control logic, and, hence its area is only about 13% smaller than the area of simple, packet-switched bus (abbreviated as pkt). In mesh, the control logic for cut-through switching is

Table 4  
Absolute logic area of the networks (in kilogates) and their relative areas

Network/ $N$	Absolute logic area (kilogates)				Relative area (on average)	
	4	16	36	64	Network	Network + PEs
Single bus (pkt)	28.7	114.8	258.4	459.1	1.14	1.02
single bus (HIBI)	25.2	100.4	225.8	401.4	1.00	1.00
Hier. bus (pkt)	28.7	158.1	373.5	675.2	1.51	1.06
Hier. bus (HIBI)	25.2	138.7	327.7	592.2	1.33	1.04
Mesh (st and fwd)	59.3	295.4	708.6	1299.0	2.92	1.21
Mesh (cut)	59.8	297.7	714.1	1309.1	2.94	1.22

Number of agents ( $N$ ) varies from 4 to 64. Area of PE is assumed to be 50 kilogates.

slightly more complex than in store-and-forward. The relative sizes are shown for networks only and for the whole system assuming a 50-kilogate area for each processing element (PE). The overhead from network considering the total system size naturally decreases as the size of processing elements increases. The area results suggest that it is possible to use wider bus, e.g. 64-bit data instead of 32-bit, and still have smaller area than with mesh. The impact of this will be described later.

#### 4.2. Simulation results

In the following simulations, the processing time  $P$  varies from 16 to 1024 cycles and the transfer length  $D$  varies from 16 to 1024 words. The extreme case  $P = 16$ ,  $D = 1024$  sets tight requirements for the communication network. This kind of communication intensive transfer patterns can be found, for example, in packet processing inside an Internet router. In addition, such case may appear when the processing elements operate with higher frequency than the network (processing time  $P$  refers to clock cycles of the network in that case). The measured average runtimes for case  $P = 16$ ,  $D = 1024$  are listed in Table 5. Times are given in thousands of cycles (kilocycles). The shortest runtimes are shown in bold. Test cases were executed several times, 30 times on average, for statistical reliability. All networks have the same data width. The run-time of case 5 is defined by the slowest individual application, case 1, when applications 1–4 are run together.

In case 1, the poor result of the packet-switched single bus is due to inefficiency of the utilized distributed round-robin arbitration during low levels of contention. HIBI, on the other hand, can transfer more data in one turn and, hence, arbitration and address transfers have less impact. Therefore, HIBI achieves the shortest execution time. In test case 2,

the differences between the networks are more apparent. The transfer times of the single bus grows very fast with system (and application) size. On the other hand, the results for hierarchical bus and mesh are quite close to each other and follow the results predicted by (1), which means that both networks are able to systematically exploit the inherent parallelism of the application. The same happens with test cases 3, 4, and 5. For these test cases, a hierarchical, packet-switched bus offers a comparable performance as 2-dimensional mesh with store-and-forward switching. However, HIBI outperforms the packet-switched bus and, similarly, cut-through mesh outperforms store-and-forward switching. Thus, in addition to topology, the switching policy has also a notable impact on the performance.

Fig. 3 shows both the simulated cycle counts as well as those estimated using Eq. (1). Three networks out of six are shown for clarity. The shapes of the estimated and the measured performance curves match rather well, which implies that Eq. (1) predicts the ratio between runtimes in many cases. However, in some cases estimates are clearly inaccurate, e.g. case 2 with mesh. This is mainly due to the implementation of TG, network contention, and inefficiencies in implemented network protocols and packetization logic that were not included in equations. Other than 1-to-1 mappings cause more contention and larger average hop counts and hence bigger estimation errors are probable. Therefore, fast simulation-based approaches for benchmarking are needed.

#### 4.3. Relative cost of the networks

Fig. 4a shows the area overhead versus average speedup curves for 36-agent networks. The speedup is estimated according to (1) and area according to

Table 5  
The runtimes in kilocycles for test cases with  $P = 16$  cycles,  $D = 1024$  words

Network/case	4 Agents					16 Agents					36 Agents					64 Agents				
	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
Single bus (pkt)	12.9	7.7	12.9	7.2	43.0	60.0	30.7	34.8	28.7	585.2	227.1	69.1	78.3	64.5	2935.7	630.8	122.9	139.3	114.7	11661.4
Single bus (HIBI)	<b>6.5</b>	<b>4.4</b>	<b>6.5</b>	5.0	24.9	23.6	18.5	18.9	19.1	257.7	71.9	31.9	42.7	38.6	1242.2	176.2	67.1	74.9	91.4	3607.7
Hier. bus (pkt)	Same as single bus (pkt)					55.0	11.3	14.5	7.3	110.5	120.0	11.5	14.5	7.3	266.4	211.1	11.5	14.5	7.3	494.3
Hier. bus (HIBI)	Same as single bus (HIBI)					<b>20.4</b>	8.9	<b>4.9</b>	5.3	97.1	<b>46.6</b>	10.5	<b>4.9</b>	5.3	308.8	<b>82.6</b>	12.9	<b>4.9</b>	5.3	332.4
Mesh (st and fwd)	15.6	7.8	15.6	3.9	43.0	62.4	7.8	15.6	3.9	115.9	140.7	7.8	15.6	4.7	308.6	249.9	7.8	15.6	3.9	267.4
Mesh (cut)	10.4	5.2	10.4	<b>2.7</b>	<b>22.9</b>	41.7	<b>5.2</b>	10.4	<b>2.7</b>	<b>74.4</b>	93.8	<b>5.2</b>	10.4	<b>4.3</b>	<b>209.8</b>	167.0	<b>5.2</b>	10.4	<b>2.7</b>	<b>196.1</b>

the number of buffers. Results are scaled so that both the smallest area and speedup are equal to one. The best approaches are in the top left corner. Hierarchical bus offers good speedup over single bus with moderate area overhead, especially HIBI that scales better than the packet-bus. Mesh offers biggest average speedup with largest area. The choice between the mesh and the hierarchical bus is, therefore, a trade-off between area cost and performance. For comparison, we define the architectural performance as the inverse of the costs:

$$\text{Performance} = \text{cost}^{-1} = (t_{\text{tot}}^{w_i} \cdot A_{\text{tot}})^{-1}. \quad (3)$$

The cost is defined as a product of runtime  $t_{\text{tot}}$  and total system area  $A_{\text{tot}}$ . System area includes both the network and all the PEs (50 kilogates each, in this case). The weighting exponent  $w_i$  can be utilized to make the runtime less ( $w_i < 1$ ) or more important ( $w_i > 1$ ) than the area. In these cases, smaller weights favor the hierarchical bus and large ones favor the mesh. Fig. 4b shows the measured performance of all networks so that the best case is scaled to 1 and execution and area have equal weights ( $w_i = 1$ ). Hierarchical HIBI and cut-through mesh offer the best trade-off. A single bus is applicable only in sequential test case 1. The utilized mapping seems to be somewhat suboptimal for 6-by-6 mesh. Although some of the 4-task groups fit in a single mesh row, others are split into two rows. This is especially notable in test case 4.

Fig. 5 shows cases where  $P = 64$  cycles and  $D$  varies between 32 and 128 words. Only the fastest networks, i.e. HIBI and cut-through mesh, are shown for clarity. The differences between networks become more apparent as the data amount  $D$  increases. Likewise, the network performance depends strongly on the application and system size. For example, HIBI is superior in cases 1 and 3 but mesh is better with cases 2 and 4, especially, with large number of agents and large data amount. In addition, Fig. 5 shows the runtime of hierarchical HIBI when its data width is doubled. It is measured by halving the amount of transferred data  $D$  in the simulation. As the data width doubles, the area of HIBI increases 86% but it is still about 15% smaller than mesh. At the same time, the runtime decreases in all test cases. The reduction varies from 20% up to 50%, being smallest in sequential cases with small  $D$ . In addition, hierarchical HIBI with wide links achieves the lowest cycle count in these cases. The average (relative) cycle counts are 1.0 (hier.

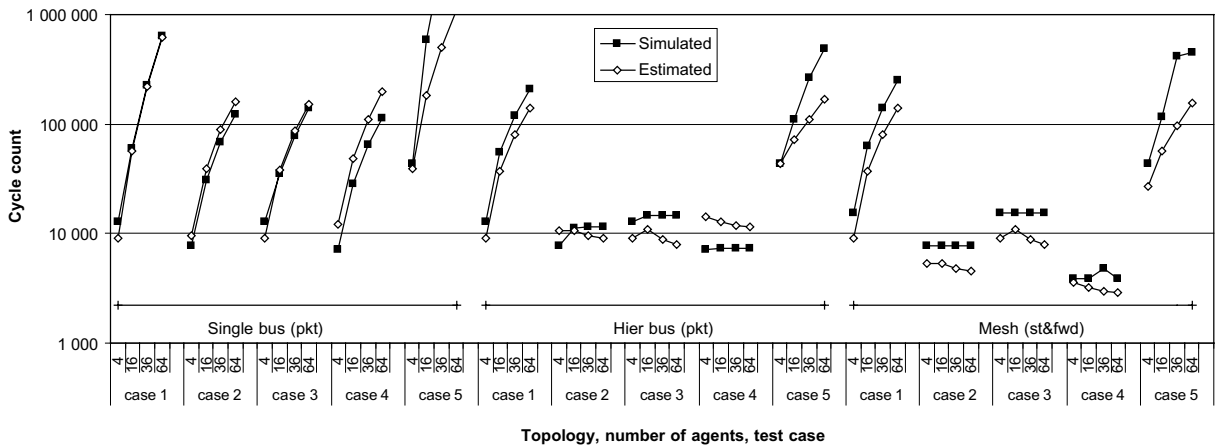


Fig. 3. Estimated and simulated cycle counts for three networks.

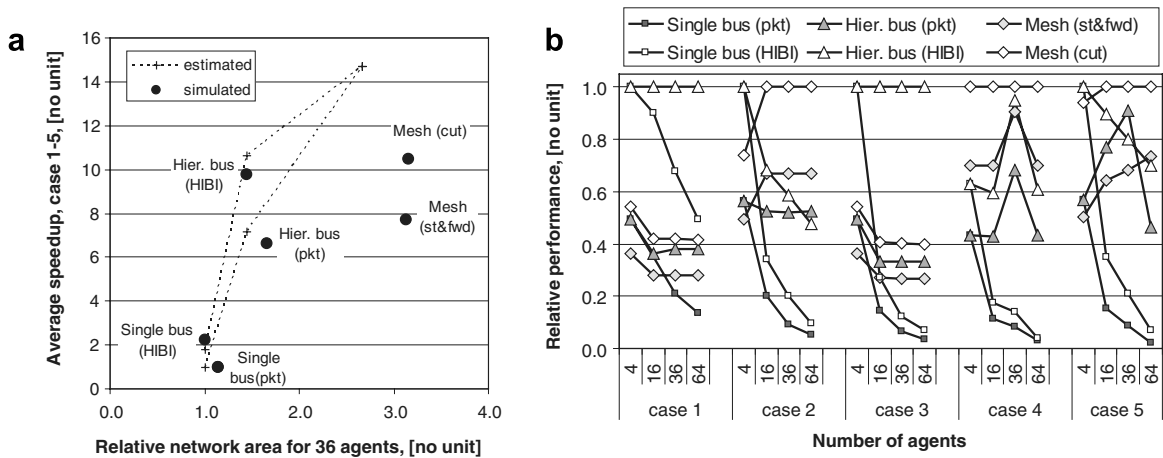


Fig. 4. Speedup and relative performance,  $P = 16$ ,  $D = 1024$ . (a) Average speedup versus area for 36 agents and (b) relative overall performance (bigger the better).

HIBI), 1.18 (mesh), and 0.67 (double wide hier. HIBI).

Finally, the differences of the networks are shown by plotting the latency versus offered load, which is a widely adopted custom [8,13,23]. The offered load is the probability that an agent injects a data word to the network on each clock cycle. The target is chosen randomly. In a basic case, all targets have a uniform probability  $1/(N-1)$ . Localized traffic is modeled by defining clusters and probabilities so that a certain fraction of transfer remains in the same cluster as the sender. In this case, the cluster size was 4 agents and locality probability was varied from 10% to 100%. For example, agents 0–3 belong

to cluster 0 agents 4–7 to cluster 1 and so on. Results are shown in Fig. 6 for 8-word transfers. Two target distributions are shown: uniform and locality probability of 60%. A single bus has the same performance regardless of locality, but hierarchical bus and mesh benefit from the increased locality. Single bus saturates quickly but offers a small latency for very small load (1–3% per agent). Hierarchical bus practically doubles the accepted traffic with uniform traffic but the real gain is achieved when transfers are localized. Mesh exhibits the largest latency (partly due to packetization logic) for small load but tolerates the largest traffic.

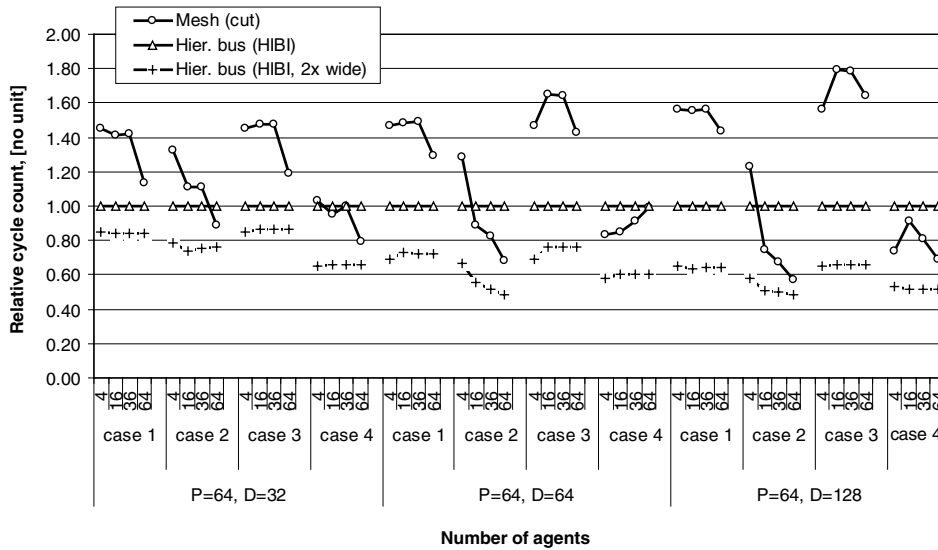


Fig. 5. The relative runtimes with 4 test cases. Processing time  $P = 64$  cycles and data amount  $D = 32, 64$  or  $128$  words.

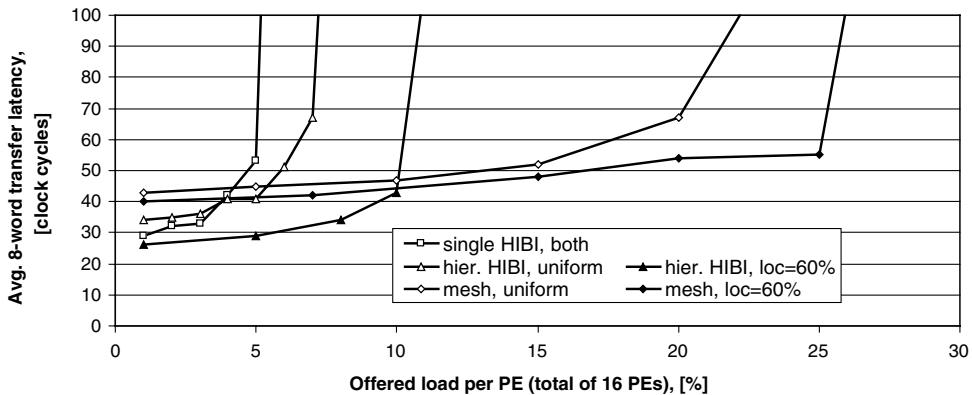


Fig. 6. Offered load vs. latency with 8-word transfers and 16 agents.

## 5. Conclusions

This paper presented a comparison of on-chip networks using synthesized models and Transaction Generator. Single bus, hierarchical bus, and 2-dimensional mesh topologies were used. Application dependence is a factor that cannot be disregarded in communication-based system design. Results with six test cases show that theoretical performance derived from the aggregate bandwidth of the network does not reflect the application runtime linearly. The presented formal equation provides reasonably accurate estimate in some cases but not in general. However, more advanced estimates considering router latency and congestion would

require rather complex equations. Therefore, fast, cycle-accurate simulation is preferred.

Many contemporary analyzes depict buses as poor fits to large systems because only the single bus is used as a reference. The presented hierarchical bus scales quite easily to large systems and provides a good area-performance trade-off while retaining many of the advantageous features of simpler bus arrangements. The hierarchical bus exhibits good runtime results with relatively small implementation area. The analyzed 2-dimensional mesh network provides the highest performance with the largest area. The architectural performance, defined as a product of area and runtime, favors the use of hierarchical bus topology. There is on-going work

for developing more test cases (both synthetic and profiled real applications), exploring different process mappings, and including other network topologies, more efficient arbitration and flow control schemes (e.g. wormhole). Furthermore, more performance and cost factors, such as energy and latency variation, will be analyzed. Such metrics are more complex to analyze formally but can be estimated through simulation.

## References

- [1] L. Benini, G. de Micheli, Networks on chips: a new SoC paradigm, *Computer* 35 (1) (2002) 70–78.
- [2] R. Ho, K.W. Mai, M.A. Horowitz, The future of wires, *Proc. IEEE* 89 (4) (2001) 490–504.
- [3] D. Sylvester, K. Keutzer, Impact of small process geometries on microarchitectures in systems on a chip, *Proc. IEEE* 89 (4) (2001) 467–489.
- [4] E. Salminen, V. Lahtinen, K. Kuusilinna, T. Hämäläinen, Overview of bus-based system-on-chip interconnections, in: *ISCAS*, vol. 2, Scottsdale, AZ, USA, May 2002, pp. 372–375.
- [5] A. Lines, Asynchronous interconnect for synchronous SoC design, *Micro* 24 (1) (2004) 32–41.
- [6] J. Liang, A. Laffely, S. Srinivasan, R. Tessier, An architecture and compiler for scalable on-chip communication, *TVLSI* 12 (7) (2004) 711–726.
- [7] D. Wiklund, S. Sathe, D. Liu, Network on chip simulations for benchmarking, in: *IWSOC*, Banff, Canada, 2004, pp. 269–274.
- [8] A. Andriahantenaina, A. Greiner, Micro-network for SoC: implementation of a 32-port SPIN network, in: *DATE*, Munich, Germany, March 2003, pp. 1128–1129.
- [9] E. Bolotin, I. Cidon, R. Ginosar, A. Kolodny, Cost considerations in network on chip, *Integration* 38 (1) (2004) 19–42.
- [10] T. Bartic, J.-Y. Mignolet, V. Nollet, T. Marescaux, D. Verkest, S. Vernalde, R. Lauwereins, Highly scalable network on chip for reconfigurable systems, in: *International Symposium on System-on-Chip*, Tampere, Finland, November 2003, pp. 78–82.
- [11] F. Moraes, N. Calazans, A. Mello, L. Möller, L. Ost, Hermes: an infrastructure for low area overhead packet-switching networks on chip, *Integration* 38 (1) (2004) 69–93.
- [12] I. Saastamoinen, M. Alho, J. Nurmi, Buffer implementation for Proteo network-on-chip, in: *ISCAS*, Bangkok, Thailand, May 2003, pp. 113–116.
- [13] W.J. Dally, B. Towles, *Principles and Practices of Interconnection Networks*, Morgan Kaufmann Publishers, 2004.
- [14] C.A. Zeferino, M.E. Kreutz, L. Carro, A.A. Susin, A study on communication issues for system-on-chip, in: *SBCCI*, Porto Alegre, Brazil, September 2002, pp. 121–126.
- [15] K. Lahiri, A. Raghunathan, S. Dey, Evaluation of the traffic-performance characteristics of system-on-chip communication architectures, in: *Conference on VLSI design*, Bangalore, India, January 2001, pp. 29–35.
- [16] M.E. Kreutz, L. Carro, C.A. Zeferino, A.A. Susin, Communication architectures for system-on-chip, in: *SBCCI*, Pirenópolis, Brazil, September 2001, pp. 14–19.
- [17] H. Zhang, M. Wan, V. George, J. Rabaey, Interconnect architecture exploration for low-energy reconfigurable single-chip DSPs, in: *Workshop on VLSI*, Orlando, Florida, USA, April 1999, pp. 2–8.
- [18] E. Salminen, T. Kangas, T.D. Hämäläinen, J. Riihimäki, Requirements for network-on-chip benchmarking, in: *Norchip*, Oulu, Finland, November 2005, pp. 82–85.
- [19] R. Thid, M. Millberg, A. Jantsch, Evaluating NoC communication backbones with simulation, in: *Norchip*, Riga, Latvia, November 2003.
- [20] T. Salminen, J.-P. Soininen, Evaluating application mapping using network simulation, in: *International Symposium on System-on-Chip*, Tampere, Finland, November 2003, pp. 27–30.
- [21] A.D. Pimentel, S. Polstra, F. Terpstra, A.W. van Halderen, J.E. Coffland, L.O. Hertzberger, Towards efficient design space exploration of heterogeneous embedded media systems, in: *LNCS*, vol. 2268, 2002, pp. 57–73.
- [22] T. Kangas, J. Riihimäki, E. Salminen, K. Kuusilinna, T.D. Hämäläinen, Using a communication generator in SoC architecture exploration, in: *International Symposium on System-on-Chip*, Tampere, Finland, November 2003, pp. 105–108.
- [23] P. Pande, C. Grecu, A.I.M. Jones, R. Saleh, Performance evaluation and design trade-offs for network-on-chip interconnect architectures, *IEEE Trans. Comput.* 54 (8) (2005) 1025–1040.
- [24] E. Salminen, T. Kangas, J. Riihimäki, V. Lahtinen, K. Kuusilinna, T.D. Hämäläinen, HIBI communication network for system-on-chip, *J. VLSI Signal Process. Syst. Signal Image Video Technol.* 43 (2) (2006) 185–205.
- [25] G. Kahn, The semantics of a simple language for parallel programming, in: *IFIP Conference*, Stockholm, Sweden, 1974, pp. 471–475.



**Erno Salminen**, MSc '01, Tampere University of Technology (TUT). Currently he is working towards his PhD degree in the Institute of Digital and Computer Systems (DCS) at TUT. His main research interests are digital systems design and communication issues in SoCs.



**Tero Kangas**, PhD '06, TUT. A research scientist at TUT 1999–2006 focusing on system architectures and SoC design methodologies in multimedia applications. In 2006, he joined Nokia Technology Platforms and is currently working there as an ASIC design specialist.



**Vesa Lahtinen**, PhD '04, TUT. A researcher at TUT 1996–2004. His main research areas were SoC design and their interconnects. Senior Research Engineer at Nokia Research Center 2004–2006 concentrating on architecture modeling and, specifically, memory architectures. Senior specialist at Nokia Technology Platforms since 2006, working on system-level design and IC consortiums.



**Jouni Riihimäki**, MSc '01, TUT. His main research interests include system-level design and simulation. Currently, he is working as a senior design engineer at the Nokia Technology Platforms.



**Kimmo Kuusilinna**, PhD '01, TUT. His main research interests include system-level design and verification, interconnection networks, and parallel memories. Currently he is working as a principal scientist at the Nokia Research Center.



**Timo D. Hämäläinen**, PhD '97, TUT. A senior research scientist and project manager at TUT 1997–2001. He was nominated to full professor at TUT/DCS in 2001. He heads the DACI research group that focuses on three main lines: wireless local area networking and wireless sensor networks, high-performance DSP/HW based video encoding, and interconnection networks with design flow tools for heterogeneous

SoC platforms.