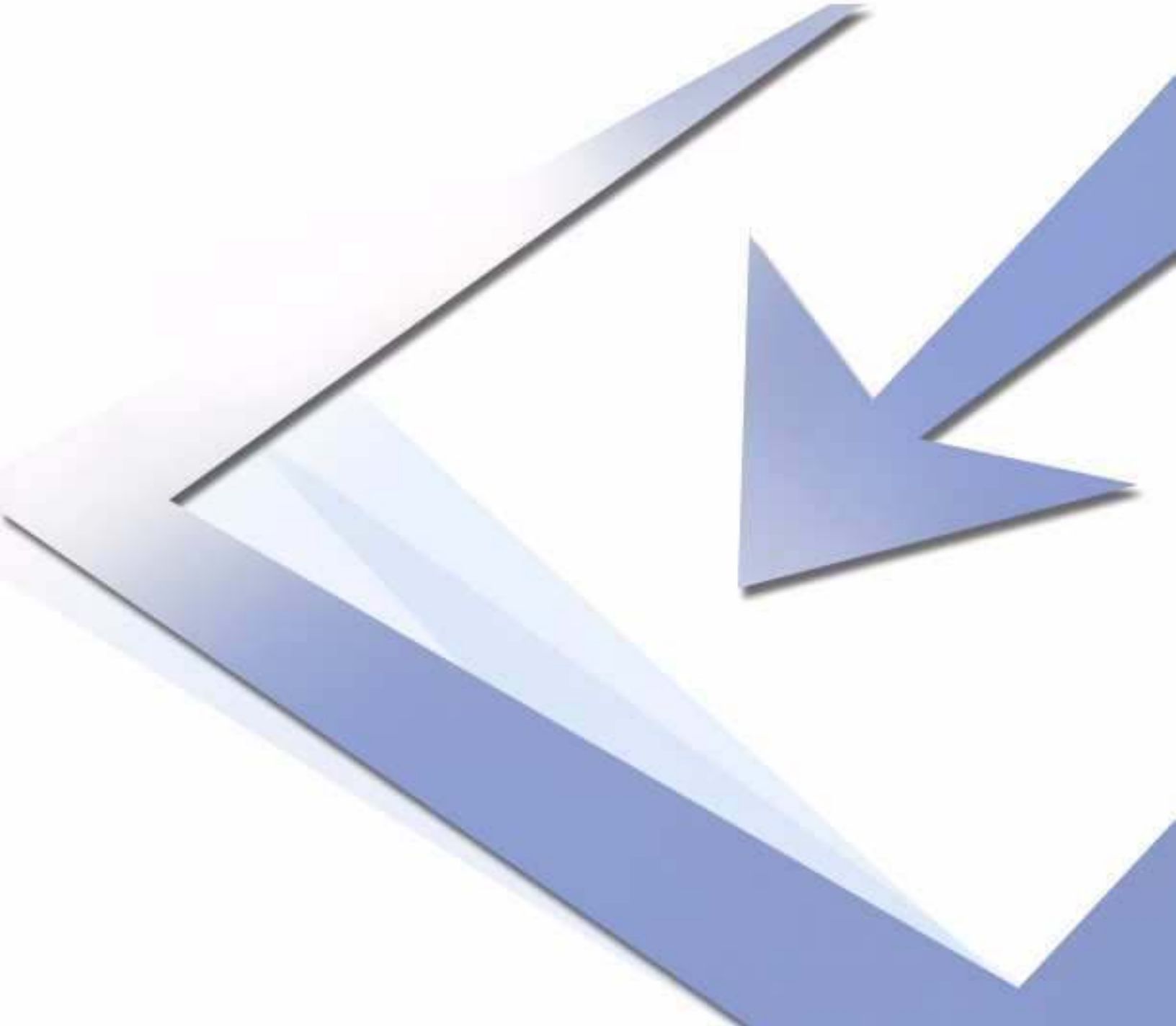
OCP International Partnership

# Network-on-Chip Benchmarking Specification
# Part 1: Application modeling and hardware description
# Version 1.0



# Network-on-chip Benchmarking Specification Part I: Application modeling and hardware description Version 1.0

Erno Salminen *, Cristian Grecu †, Timo D. Hämäläinen *, and André Ivanov †
* Tampere University of Technology, P.O. Box 553, FIN-33101 Tampere, Finland
† University of British Columbia, 2332 Main Mall, Vancouver, Canada
Modified 4.4.2008

**Abstract**

Measuring and comparing performance, cost, and other features of advanced communication architectures for complex multicore/multiprocessor systems on chip is a significant challenge which has hardly been addressed so far. This article presents a modeling methodology for applications running on multicore systems and defines an XML format for documenting and distributing network-on-chip benchmarks. It defines a black-box view of the processing elements that discloses only the computational aspects that are relevant in interacting with the on chip data transport mechanism.

Keywords: network-on-chip, benchmarking, comparison, modeling, XML

ACKNOWLEDGEMENTS

The authors would like to acknowledge the contribution of the other members of the Network-on-Chip Benchmarking Workgroup at OCP-IP: Axel Jantsch and Zhonghai Lu (Royal Institute of Technology - Sweden), Partha Pande (Washington State University), Radu Marculescu and Umit Ogras (Carnegie Mellon University), Pascal Chauvet (Sonics), and Yasuhiko Kurosawa (Toshiba).

Valuable comments and discussions were provided by members of other OCP-IP working groups: Steven Mcmaster (Synopsys), Vesa Lahtinen (Nokia), Mark Burton (GreenSocs), Srinivasan Krishnan (Sonics), and Jeff Deutch (Texas Instruments). Special thanks are due to OCP-IP President, Ian Mackintosh, for his continuous and active support of this initiative.

CONTENTS

## I. INTRODUCTION

Network-on-Chip (NoC) paradigm [1][2] brings the techniques developed for macro-scale, multi-hop networks into a chip. The major goal of communication-centric design methodologies and NoC is to achieve greater design productivity and performance by handling the increasing parallelism, manufacturing complexity, wiring problems, and reliability.

No optimal NoC exists in general case and a brute-force search is impossible due to vast design space. Benchmarking, however, allows us to identify the most promising solutions which are then selected for detailed and time consuming analysis. This reduces the design time notably once the major characteristics and requirements of the system are known. Benchmarking must be done with care and by following strict scientific principles both in measurements and in reporting. A recent survey on state-of-the-art NoC comparisons [3] together with [4][5] clearly motivated the need for a common benchmarking strategy and test cases.

The main concepts for benchmarking NoCs in a systematic and comparable way are currently addressed by an OCP-IP workgroup [6]. This complements that work by presenting a common modeling style and file formats which are necessary for documenting and distribution of benchmark cases. To our knowledge, this is the first attempt to capture a common, disciplined benchmarking methodology targeting network-on-chip. The benchmarks will be delivered as XML (eXtensible Markup Language) files.

This paper is structured as follows. Section III presents the overall view. Sections IV-VII introduce the four main parts of the application models followed by conclusions in Section VIII. A short example of the utilized XML is given in Section IX.

## II. EXTENSIBLE MARKUP LANGUAGE (XML) AND SPIRIT

eXtensible Markup Language (XML) is a standard for creating markup languages which describe the structure of data [9][10]. The set of elements is not fixed as in HTML. Instead, users can define their own tags. The primary purpose of the XML is to facilitate the sharing of structured data across different information systems. XML files are plain text files, which are less restrictive than other proprietary document formats. An XML *schema* describes the type of the XML document, typically expressed in terms of constraints on the structure and content of documents, above and beyond the basic constraints imposed by XML itself. The strict syntax and parsing requirements, in the form of schema, make the parsing algorithms extremely simple, efficient, and consistent. A schema will be provided as part of the NoC benchmark set to validate the NoC XML files (see www.ocpip.org for further details).

The NoC benchmarking XML is somewhat similar to SPIRIT consortium's IP-XACT system model [11] but has a bit different scope and higher abstraction level. IP-XACT (versions 1.0 and 1.2) is aimed for documenting the register transfer level (RTL) interfaces of the intellectual property (IP) components whereas the NoC XML describes the application model and its mapping on the hardware architecture. Furthermore, NoC XML does not specify individual signals or pins but sockets. A socket groups together all the data and control signals present in certain interface and specifies their timing and other relations.

## III. BASIC STRUCTURE AND MODELING CONCEPTS

Fig. 1 depicts the proposed NoC system model. Some of the basic concepts of the system model and a very similar format were previously presented in the Koski design automation tool set [7][8]. The model and the corresponding XML description are divided into four main sections:

1) *Application* defines the computation and communication load
2) *Mapping* binds the application tasks to the resources
3) *Platform* defines the resources and the NoC interconnecting them
4) *Measurement* section defines how to perform the evaluation, for example metrics and simulation length.

Fig. 1 shows a simple task in network consisting of six tasks ($A-F$) and two triggering events, which trigger tasks $A$ and $D$. The tasks are grouped into five groups ($I-V$) that are mapped to four PEs ($PE0-PE3$). The data amounts in bytes are expressed beside each edge. Assuming that the events in this application are periodical and their time interval is set properly, all the PEs 0-3 can execute tasks simultaneously.

Separation to distinct parts is necessary to handle complex architectures and applications [12]. Orthogonality allows exercising or modifying one of the components, while keeping the rest at their previous configuration. Thus, the mapping, for example, may be varied without touching the application or hardware models. Similarly, one can describe the particular NoC once but change the application when needed.

The benchmarks set covers the three uppermost sections of Fig. 1 (application, mapping, computation resources) and measurement settings but leave the network definition to the NoC designer/evaluator. The XML is given as input to a traffic generator that will be used during the simulation. The traffic generator will inject/eject traffic to/from the NoC, calculate statistics (PE utilization, data latencies etc.), and detect transmission errors (corrupted, dropped, out-of-order data etc.). Message-passing communication paradigm is assumed at this stage of research.

Fig. 2 shows the major tags and their relations in the proposed XML description. The top level tag is called system, it includes four tags under it, the first of those, application, includes two and so on. The numbers show the minimum number of

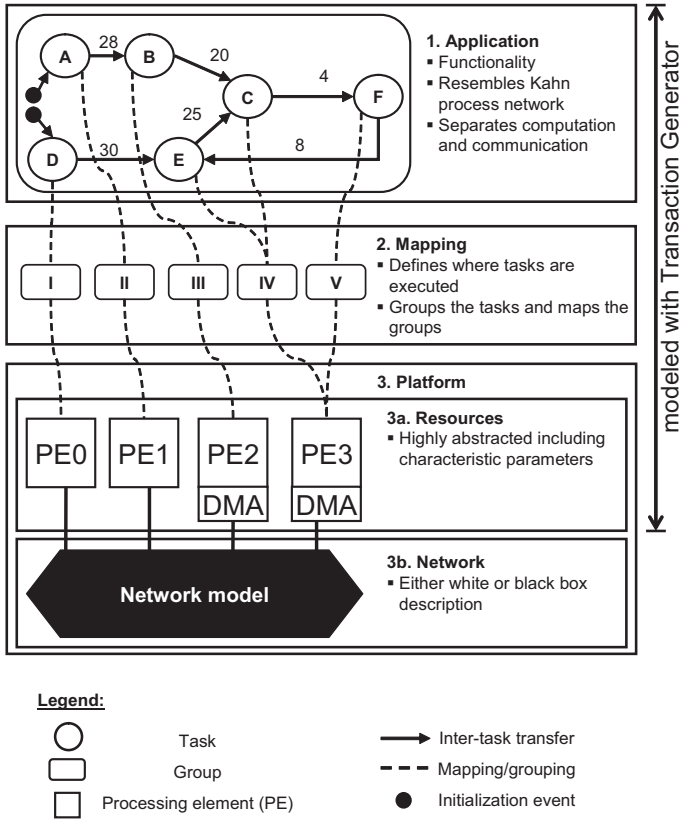



Fig. 1. Conceptual view of the utilized system model.

occurrences of each tag. A number followed by the plus sign (+) denotes that multiple instances of the same tag are allowed. In those cases, the upper bound is case-dependent (if any).

Next we introduce the major sections of the XML model.

## IV. Application model

An application includes a task graph that models the computational load as well as the induced communication. In the basic case, there is only one task graph. Model includes dependencies, timing, destination, and size of data transfers. In a task graph:

1) *task* nodes model the computation. The execution time is derived from the task's operation count and the properties of the PE executing that task No actual computation is performed during the simulation, only the external behavior of application tasks is modeled. (Actual computation is possible in [7], but not necessary at the first stage of NoC benchmarking.)
2) *connections* between nodes carry out the communication. Tasks nodes communicate via connection channels (directed edges) that carry the data tokens between tasks. Channels are attached to the tasks' ports. Bidirectional ports are not allowed.
3) *trigger events* generate stimulus to the tasks
4) *path definitions* are used for measurements, especially when real-time constraints are present.

Model of Computation (MoC) is similar to Kahn Process Network (KPN), where vertices represent computation tasks and edges represent communication channels [7]. Chosen model is reactive due to dependencies. For example, when a certain transfer is delayed, all the tasks (and transfers) depending on that transfer are also delayed. The task set is static and no tasks are spawn during execution.

Application model may include several task graphs (smaller applications) in order to model multitasking. Tasks in different graphs may be connected. Even if graphs are disjoint (no communication between graphs) they can still affect each other's execution times as they may compete for the shared resources, i.e. PEs and NoC. Some tasks may model memory accesses instead of computation.

### A. Application tasks

Tasks are the primary means of expressing the communication and computation load. Fig. 3 shows the tags used for describing the tasks. Tasks communicate via unidirectional ports. A task is triggered for execution according to a condition that depends

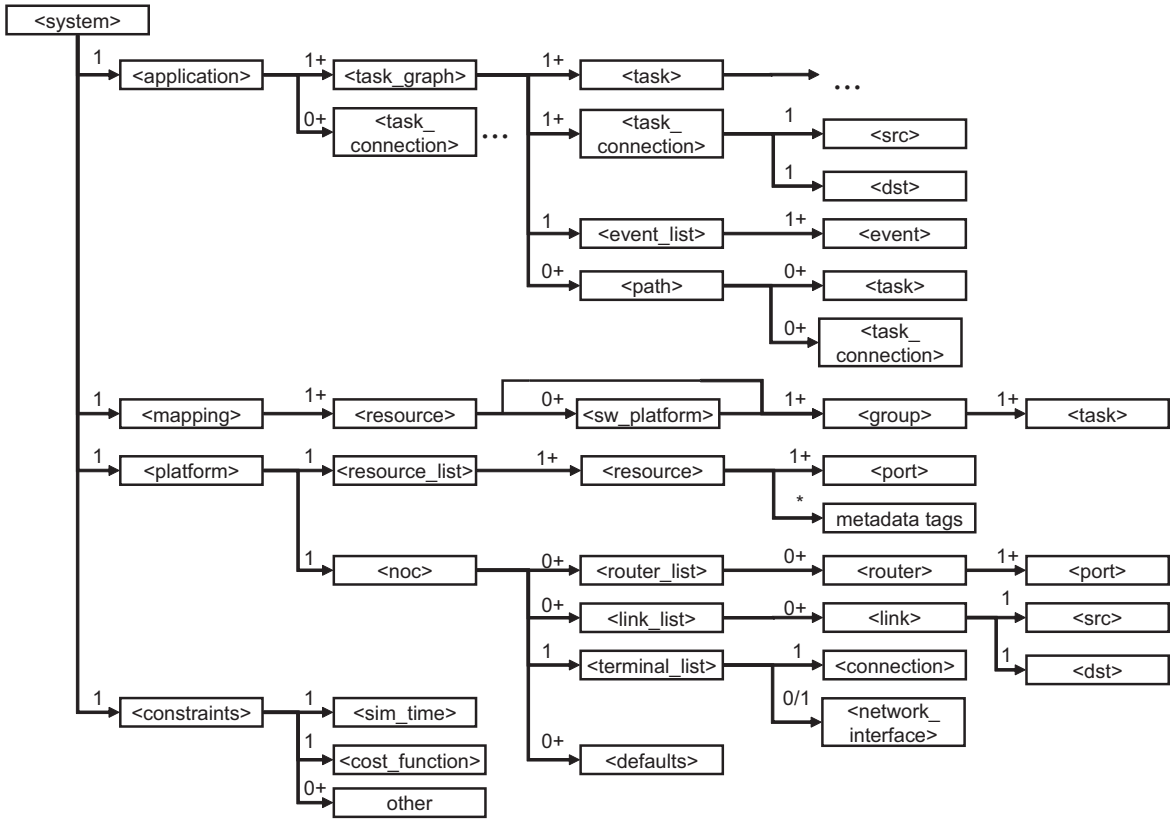



Fig. 2. Major tags in XML system model. The numbers show the minimum number of occurrences of each tag. Each of the four major sections occurs exactly once. Task description is shown in Fig. 3.

on the received data tokens and possibly on the internal state of the task. A task may include several behaviors but exactly one is executed at a time.

Triggering conditions have two variables: input ports and the execution history of the task. The number of ports in the task or trigger condition is not limited. Hence, all inputs of the task can be present, if needed. There are two types of dependencies on multiple input ports

1) *AND*: triggered when there is data in all inputs,
2) *OR*: triggered when any of the input receives a token.

The OR dependency distinguishes the model from the traditional KPN. The history means here simply the count of how many times the task has been executed. For example, a task may have a different behavior on the first execution but after that the same on all other executions. Another example could be that a task performs function $A$ on every even execution count and function $B$ on every odd. In the extreme case, the task behavior is different on each execution. This is suitable for capturing a trace for highly varying tasks for which the average values are misleading.

Each execution behavior is characterized by three elements:

1) the operation count,
2) data amount to be sent (in bytes) and the output ports where the data will be sent, and
3) the next state of the task after execution.

The operation count and data amount are expressed with either a statistical distribution (uniform, normal, or Poisson), or as a polynomial function, which depends on the received data amount (a constant value is subset of the latter). Polynomial and statistical are mutually exclusive choices and hence denoted with (i) and (ii) in Fig. 3. However, the choice is done per task; certain tasks may have polynomial count/amount values and others statistical ones.

Each output is assigned a certain probability. The same output is always chosen if its probability is equal to $1.0$, i.e. in $100\%$ of cases. Other probability values allow compacting the model as less triggering conditions are needed. There is one send tag for each destination.

Few possible example behaviors with different triggering conditions are

- Always perform the same action (same number of operations, same amount of output data, same destinations) irrespective of execution count or input port.

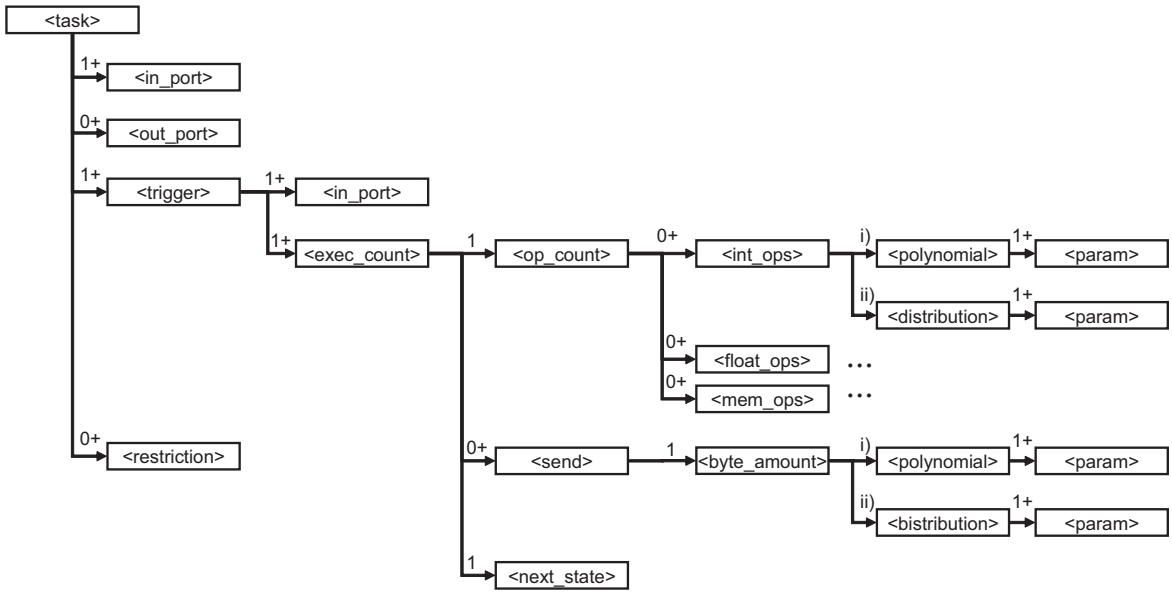
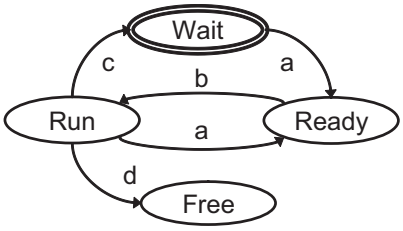



Fig. 3. XML tags for describing the tasks.




a: The task has all the data for the next execution
b: Scheduler selected the task for execution
c: Fully executed but has not all data for the next execution
d: Fully executed and is not dependent on any data anymore

Fig. 4. The possible states of an application task during simulation.

- Operation count and data amount depend on the received data amount but not on the execution count; destination(s) are always the same.
- Select the output according to the input port similarly on every execution; however, operation count and data amount may be constants, polynomial, or statistical.
- All parameters depend on both the input port list and execution count.

*1) Task state machine :* During simulation, the internal state of a task is expressed with an execution counter and a status variable. Fig. 4 illustrates the scheduling state machine of a single task. Tasks start in state “wait”. Once it receives enough data tokens and triggering condition is fulfilled, it will become ready for execution. Scheduling policy of the PE defines which task from the ready list is selected for execution (edge b). Parameters related to scheduling are defined in the mapping part. After execution, task’s next state is usually set again to ”wait”. It may also be ”free” (as indicated in Fig. 4 which means that the task won’t be executed anymore. The execution counter is incremented every time when the task leaves the state ”Run” (pre-emption necessitates detecting the completion of execution and not just state change).

Every task may have other restrictions and settings, for example, can a task be pre-empted or to which PEs is may be mapped. Hence, these affect the mapping and scheduling possibilities. This section is included mainly for future extensions.

### *B. Connections*

The connections simply define the structure of the task graph: which tasks communicate with each other. Their only parameters include the source and destination ports of the tasks.

### C. Triggering events

Triggering events model the environment and usually at least one event is required to start the task graph. In other words, they generate the input data stream that is ”processed” by the application.

Their behavior is similar to that of timers. Events are special nodes because they are not mapped to any PE, and hence do not reserve computational resources. However, they use regular connections to transmit their data to the tasks. Events are either *periodic* or *one-shot*. Their data emission may have a probability less than 100% if needed. This means that an event does not always send data at all although it is triggered. They do not consume ”CPU-time” of the resources or NoC capacity.

As mentioned, all tasks are initially in state ”wait” and, therefore, at least one event is needed to trigger the execution. Currently, events do not have input ports so they cannot react to the application’s outputs.

### D. Real-time constraints and paths

A task graph can also include real-time constraints, for example, computation deadlines with tasks and communication deadlines with connections. In addition to single tasks (node) or connections (edges), *paths* that constitute several tasks may be defined.

A path is a list of tasks and/or connections that is used for benchmarking. For example, path $p$ includes tasks $A$, $B$, and $C$ (in this order) and this path has a real-time constraint of $2\ ms$. It is measured from the triggering of task $A$ to the completion of task $C$. The transaction generator will monitor is the deadline always met and report violations. In addition, the average runtime of the path may be also used in to define the cost function.

There are multiple choices to start event respect to which the real-time constraints are given. The time may be measured starting from the time instant when

- triggering condition $X$ occurs,
- current task received all its input data,
- previous task is completed,
- current task last completed.

## V. Mapping model

Mapping model defines on a per-PE basis where the tasks are executed. Mapping is performed in two stages: grouping of tasks together, and mapping the groups to resources. A less expressive mapping could be done without the groups, but the chosen method allows more possibilities (including the simple mapping). If uncertain, users can have just one task per group. The main ideas in grouping are to model operating system threads and to restrict mapping exploration. Both tasks and groups may have parameters related to scheduling, such as priorities.

Group contents may be modified by automated design automation tools if allowed (*mutable=”yes”*) or they can be moved elsewhere (*movable=”yes”*). The former restriction can be also done for a PE and the latter for a task. Although the application is fixed in each benchmark, there are various options:

- assign fixed mapping for a NoC of a certain size, say 16 terminals. Designers can easily vary NoC parameters without modifying other parts of the XML description.
- assign fixed grouping of tasks, and let the designer map them to his/her NoC freely. Note that the number of NoC terminals can be smaller than the number of groups.
- as previous, but without groups. This option gives full freedom in mapping.

The tag software platform is included for the sake of future extensions.

## VI. Platform model

Platform has two parts: resources and NoC. The resources will be defined by the benchmark set and the NoC by its designer/evaluator.

### A. Resource model

Computation architecture is modeled on a very coarse level. The PEs are characterized with few parameters, such as operations-per-cycle, silicon area, power consumption. In [3], these values are stored in separate PE library but here they are directly in the XML for simplicity. For each resource the benchmark defines

1) type: processing element (PE) or storage
2) NoC terminal where it is connected to
3) operating frequency, number of operations per cycle, area (mm2 or kilogates, aspect ratio), active and idle power consumption
4) inclusion of DMA controller
5) the associated communication overheads.

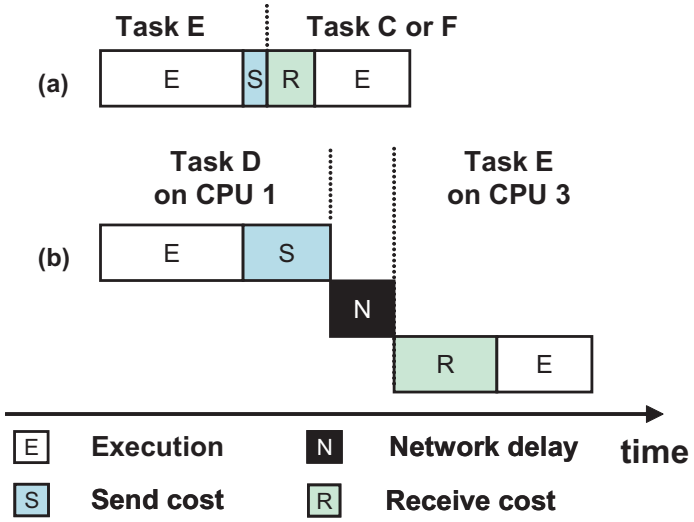



Fig. 5. Different communication costs between two tasks when a) tasks are on the same PE or b) on different PEs. The symbols correspond to Fig. 1.

The PE model performs the scheduling of tasks. The scheduling choices are FIFO (default choice), preemptive/nonpreemptive static priority, and round-robin. Priorities are set at compile-time. When a task is mapped on a PE and scheduled for execution, the simulation engine calculates the associated runtime. The cycle count becomes:

$$N_{cycles,i,pe} = N_{ops,i}/IPC_{pe}, \tag{1}$$

where $N_{ops,i}$ is the operation count of the task $i$, and $IPC_{pe}$ defines how many operations the PE can execute in one clock cycle. The runtime of task $i$ on that PE is then:

$$t_{i,pe} = N_{cycles,i,pe}/f_{pe}. \tag{2}$$

When the task has finished its execution, it emits data token(s). A direct memory access (DMA) controller allows simultaneous computation and communication which means sending or receiving data while executing application tasks. The PE model checks the destination of the transfers in order selects whether the data is transmitted to the NoC or is the communication internal to PE.

This defines which PE-specific communication costs are applied [13]. The communication overhead has the form

$$t_{comm,pe} = a + bx, \tag{3}$$

where $a$ is constant cost (cycles), $x$ is the data amount in bytes, and $b$ is the cycles per byte. Values of $a$ and $b$ are defined separately for each PE type and its software platform. They are fixed in the benchmark's XML document.

Time overhead is smallest when the source and destination tasks are in the same group (thread). A different cost is used when the destination is in another group but still on the same PE. The largest overhead occurs when the destination task is mapped on a different PE. In that case, the delay induced by the network will be added to $t_{comm,pe}$. However, note that the network delay is not static, unlike $a$ and $b$, but it is determined at the simulation time.

For example in Fig. 1, communication between the tasks $E$ and $C$ is type *intragroup*, between $C$ and $F$ *intergroup* and between $C$ and any other task *inter-PE*. The two latter can possibly cause a context switch at the receiver PE. Only the inter-PE communication is passed via network model.

Fig. 5 presents two examples of inter-task communication. In a), the communication occurs between two tasks in the same PE whereas b) presents inter-PE communication. In both cases, the execution times of the tasks are the same. The network delay is naturally not present in a), but otherwise the communication procedure is the same as in b). In addition, both send and receive costs are shorter in a), because the data does not have to be prepared for the network transaction.

### *B. Network model*

This section presents an XML description of a NoC, although the NoC is not usually defined in the benchmark. The purpose is to promote a consistent way of documenting the experiments and also to enhance inter-operability of EDA tools. The main emphasis is on documenting the topology and basic parameters. NoC is here defined by its:

1) Terminals which may include network interfaces
2) Topology, i.e. routers and links
3) Parameters (optional part).

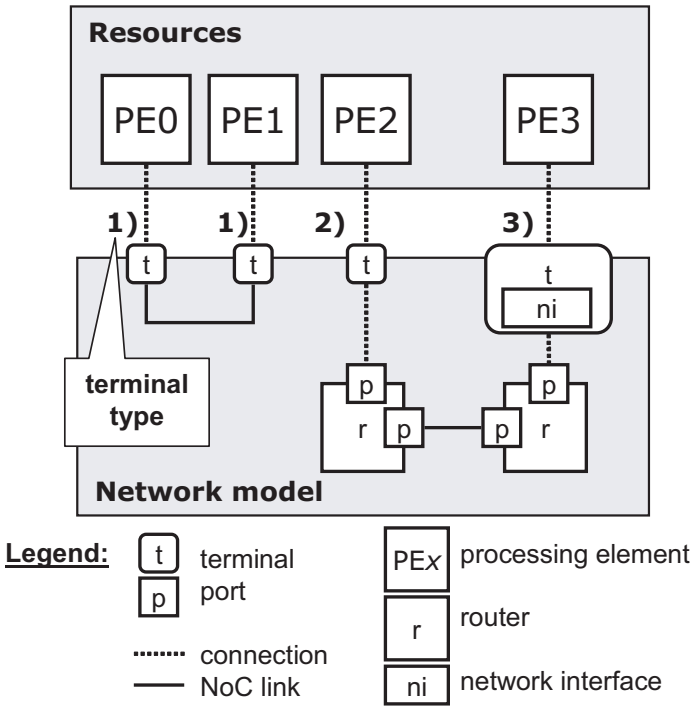



Fig. 6. Three types of supported NoC terminals.

*1) Terminals and Network Interfaces:* PEs are bound to NoC terminals which are the ”public interface” of a NoC. This provides a generic way to bind PEs without exposing the internals of the NoC. It depends on the NoC internals how the terminals are connected inside the network. In contrary, connecting PEs directly to router ports is a network-specific way and not reusable.

Fig. 6 illustrates three different terminal types.

1) On the left, there is a direct point-to-point link between $PE0$ and $PE1$. The corresponding terminals are simply the two ends of the link. Of course, the PEs must have directly compatible interfaces.
2) In the middle, $PE2$ communicates via router. Each terminal is associated with certain router port. PEs must use the communication protocol (signals, their timing, and packet structure) as the router ports. Hence, the terminals may be conceptual, i.e. just aliases to certain router ports, which do not require any logic in HW implementation.
3) On the right, there is a network interface (NI) in addition to the router. This is the most generic and probably the most common option. The interface does conversion between communication protocols. For example, $PE3$ could have an OCP-IP compliant interface but the router could use a different, proprietary protocol and expects exactly 8-word long packets with 2-word headers. The NI executes packetization (adding/removing headers), checks integrity, and possibly reordering of packets.

A terminal may include network interface logic, as shown on the right, but it is not always necessary. One might consider defining NI together with the resource. Here, they are within the NoC because that part is commonly left undefined in benchmark and completed by the NoC designer or evaluator.

One might consider also point-to-point links with network interfaces, although this option seems rare. It is left out from the figure but its behavior is easy to understand with the previous examples.

*2) NoC topology:* The NoC may be instantiated as a *white-box* model that defines all necessary topological details, or as a *black-box* that defines only the terminals and parameters in XML.

Fig. 6 illustrates also the internal structure of a NoC: there are links and routers. Links connect the ports of the routers which can be uni- or bidirectional. Bidirectional ports and links simplify XML description in many cases although in practice they are likely realized as a pair of unidirectional ports/links.

Each router includes the following basic parameters

1) varying number of ports and their directions
2) flit width and optionally also frequency
3) buffer capacity (number of flits per buffer) per port.
4) number of virtual channels per port
5) minimum latency for forwarding the packet header

Each link defines

1) the end points, each defining a router and its port
2) number of wires in one direction
3) pipeline depth, default is 0 which means no pipelining
4) operating frequency (optional).

The connections between routers and PEs are included in terminal description. Occasionally, the link or router list may be empty.

*3) (Default) NoC Parameters:* Default values can be defined for any parameters, such as data width, to avoid repeating them for every single router or link instance. However, they can be overridden on a per-component basis.

The most common parameters, see the previous subsection, have specific tags but it is possible to include additional, arbitrary parameters as *name-value* -pairs. This supports using XML as input for proprietary configuration and synthesis tools. For example, specific arbitration, routing, queue management options may be given this way. The XML file may define arbitrary parameters for the NoC or its sub-lists.

Another use case is when the NoC is used as black-box and the actual structure is described separately in VHDL files (or similar) instead of XML. The compilation and synthesis parameters are extracted from the XML and passed to corresponding EDA tools. This allows instantiating third-party NoCs without knowledge of their internals while allowing parameterization.

The white-box NoC model can represent arbitrary network topologies but might be somewhat awkward for a large, say 100-node, NoC. The black-box approach is suitable in such case, especially for regular topologies. The designer/evaluator must only define the name of the library component (e.g. 2-D mesh), network dimensions ($x = 10$, $y = 10$), terminal list ($0 - 99$), and possibly some other parameters. Effort is minimized as there is no need to describe individually all the routers and links. There is no need to describe individually all the 100 routers, all 360 links, connect links between west_out(x,y) to east_in(x+1,y) (except on the edges!), and so on. The detailed XML description, such as IP-XACT, may be eventually generated automatically based on the above parameters, if desired.

## VII. Measurement constraints

This section includes restrictions related to the simulation and benchmarking. The simulation time may be defined as a constant value, for example 200 miliseconds. However, adequate absolute time is hard to come up with in general case and hence there are other, dynamic conditions for the simulation length:

- total of $B$ bytes has been injected/ejected
- total count of all tasks' executions equals $N$
- task $X$ has been executed $N$ times
- communication edge $X$ has been executed (used) $N$ times
- path $P$ has been traversed $N$ times (paths are defined inside a task graph)
- same as previous but for several tasks/edges/paths are measured and average or maximum time will be used.

These conditions ensure that the evaluated system reaches steady state and that enough data has been transferred to obtain reliable performance results.

This section defines also the criteria that must be measured and reported and at least one cost function. Cost function combines various metrics into single value (smaller the better) that can be used for direct comparison and ranking of approaches. Example metrics are silicon area, power or energy consumption, simulation time, path time, execution count of a task, sum of all tasks' execution counts, deadline violations, and packet latencies. The deadline related information defined separately in the application description.

Basic arithmetic operations (sum, subtract, multiply, divide, power) and weights are applied to the metrics. The delivered benchmark set fixes a certain minimum set of criteria and cost functions. Researchers are, of course, encouraged to perform other complementary measurements and the same XML format can be used for reporting those studies also. The workgroup is currently preparing separate rules and guidelines for conducting the NoC studies in order to avoid ambiguities and loopholes.

## VIII. Conclusions

This paper presents a common modeling style and XML file format for performing and delivering network-on-chip benchmarks. They capture the necessary aspects in an easy to manage format. Authors believe that common, publicly available benchmarks in the form presented here offer a valuable contribution to the current state-of-the-art in NoC research. The goal is to promote disciplined evaluation, sharing of ideas and test cases, as well as to promote healthy competition within NoC and MP-SoC communities.

In the fufure the workgroup will prepare general guidelines and instructions for performing NoC evaluations as well as publish the actual benchmarking cases. Since this is work in progress, we encourage the readers to contact the authors and other workgroup members in any relevant issue or improvement suggestion.

## REFERENCES

[1] W. Dally and B. Towles, Route packets, not wires: on-chip interconnection networks, in DAC, 2001, pp. 684-689.
[2] Tobias Bjerregaard and Shankar Mahadevan, A Survey of Research and Practices of Network-on-Chip, ACM Computing Surveys, Vol. 38, Iss. 1, article No. 1, 2006.
[3] Erno Salminen, Ari Kulmala, Timo D. Hämäläinen, On Network-on-chip comparison, Euromicro conf. on Digital System Design, Lübeck, Germany, Aug. 2007, pp. 503-510.
[4] J.D. Owens, W.J. Dally, R. Ho, D.N. Jayasimha, S.W. Keckler, Li-Shiuan Peh, Research Challenges for On-Chip Interconnection Networks, IEEE Micro, Vol. 27, Iss. 5, Sept.-Oct. 2007, pp. 96 - 108.
[5] Umit Y. Ogras, Jingcao Hu, Radu Marculescu, Key research problems in NoC design: a holistic perspective, CODES 2005, pp .69-75.
[6] Cristian Grecu, André Ivanov, Partha Pratim Pande, Axel Jantsch, Erno Salminen, Umit Ogras, Radu Marculescu, ”An Initiative towards Open Network-on-Chip”, white paper, [online]:http://www.ocpip.org/socket/whitepapers/NoC-Benchmarks-WhitePaper-15.pdf, February 20, 2007, 16 pages, OCP-IP.
[7] Tero Kangas, Jouni Riihimäki, Erno Salminen, Kimmo Kuusilinna, Timo D. Hämäläinen, ”Using a Communication Generator in SoC Architecture Exploration”, International Symposium on System-on-Chip, Tampere, Finland, Nov. 2003, pp. 105-108.
[8] Tero Kangas, Petri Kukkala, Heikki Orsila, Erno Salminen, Marko Hännikäinen, Timo D. Hämäläinen, Jouni Riihimäki, Kimmo Kuusilinna, ”UML-based Multi-Processor SoC Design Framework”, Transactions on Embedded Computing Systems, May 2006, Vol.5, Issue 2, pp. 281-320
[9] W3C, ”Extensible Markup Language (XML)” , [online] http://www.w3.org/XML/.
[10] ”XML”, in Wikipedia, The Free Encyclopedia, [online] http://en.wikipedia.org/wiki/XML
[11] SPIRIT Schema Working Group, ”SPIRIT-User Guide v1.0”, Dec. 2004, [online] http://www.spiritconsortium.org/.
[12] Alberto L. Sangiovanni-Vincentelli. ”Quo Vadis SLD: Reasoning about Trends and Challenges of System-Level Design”. Proceedings of the IEEE, Vol. 95, Iss. 3, Mar. 2007, pp. 467-506.
[13] Kalle Holma, Mikko Setälä, Erno Salminen, Timo D. Hämäläinen, ”Evaluating the Model Accuracy in Automated Design Space Exploration”, Euromicro conf. on Digital System Design, Lübeck, Germany, Aug. 2007, pp. 173-180.

**Erno Salminen**, MSc ’01, works as a research scientist at Tampere University of Technology (TUT) towards PhD degree. He is a member of DACI research group and OCP-IP NoC benchmarking workgroup. His research interests along the NoC include multiprocessor SoCs, system-level design, and digital design in general.

**Cristian Grecu**, MSc ’03, is Ph.D. Candidate at University of British Columbia, expected graduation on May 2008. His research interests include design of fault-tolerant massive multi-core integrated systems, as well as post-manufacturing test methods and the performance of network-on-chip communication structures. He is a member of OCP-IP NoC benchmarking workgroup

**Timo D. Hämäläinen**, MSc ’93, PhD ’97, works as a full professor at TUT. 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.

**André Ivanov**, professor at University of British Columbia and IEEE Fellow. His primary research interests are in the area of integrated-circuit (IC) testing, design for testability, and built-in self-test for digital, analog, and mixed-signal circuits and systems-on-chip (SoCs).

## IX. APPENDIX: XML EXAMPLE

This short example aims to show the basic properties of the proposed XML format. The symbols correspond to Fig. 1 and Fig. 6.

```
<?xml version='1.0'?>
<!DOCTYPE system_description>

<system_description xmlns:noc="http://www.ocpip.org/namespace/noc" xsi:SchemaLocation="../xsm_content_model.xsd">

  <application>
   <task_graph id="Example_application">
        <task id="A" name="filter" class="general" >
         <in_port port_id="0"/>
         <out_port port_id="1"/>

         <trigger dependence_type="or">
          <in_port_ref value="0" />
           <exec_count>
             <op_count>
               <int_ops>
                  <distribution>
                     <uniform min="100" max="150"/>
                  </ distribution >
               </int_ops>
             </op_count>
             <send out_port_ref="1" prob="1.00">
                <byte_amount>
                 <polynomial>
                    <param value="28" exp="0"/>
                 </polynomial>
               </byte_amount>
             </send>
             <next_state value="READY"/>
           </exec_count>
         </trigger>
         <restriction> For future enhancements </restriction>
        </task>
        <task id="B">…</task>...<task id="F">… </task>

       <task_connection>  <!-- Event's data goes to task A -->
          <src task_ref="e0" port_ref="0"/>
          <dst task_ref="A" port_ref="0"/>
        </task_connection>
        <task_connection> … </task_connection>

        <event_list>
          <event id="e0" name="timer_1" out_port_id="0" amount="1"
                      trigger_type="one-shot" prob="1" time_sec="5.0e-006"/>
        </event_list>

        <path id="p1" name="upper_branch">
          <task> A </>
          <task> B </>
          <task> C </>
        </path>
   </task_graph>
```

```
  <!-- Several task graphs may be combined to form a single, heterogeneous application -->
  <task_graph> Another application model  </task_graph>

  <connection>
        <src tg_ref=”Example_application” task_ref=“F” port_ref=“5”>
        <dst tg_ref=“other_tg_identifier” task_ref=“start” port_ref=“4”>
  </connection>
</application>


<mapping>
  <resource ref=“PE0” contents=“mutable”>
    <group id=“I” position=“movable” contents=“mutable“>
      <task ref=“D” position=“movable“/>
    </group>
  </resource>
  ...
  <resource ref=“PE3” contents=“mutable”>
    <group id=”IV” position=“movable” contents=“mutable“>
          <task ref=”E” position=”movable”/>
          <task ref=”C” position=”movable”/>
    </group>
    <group id=”V” position=“movable” contents=“mutable“>
          <task ref=”F” position=”movable”/>
    </group>
  </resource>
  <Other resources here...>
</mapping>


<platform>
  <resource_list>
    <defaults>
      <frequency MHz=“100” />
    </defaults>
    <resource id=“PE0” type=“pe” >
      <port id=“portX” terminal_ref=“0” />
      <frequency MHz=“200” />
      <performance ops_per_cycle=”1.0” />
      <area kilogates=”50e3” ratio_y_per_x=”1.0”/>
      <dma activated=“yes” />
      <sw_platform  context_switch_cycles=”500” ... />
    </resource>
    <Other resources here...>
  </resource_list>

  <noc>
    <router_list>
      <defaults>
          <data_width bits=“32” />
          <buff_depth flits=“5” />
          <n_virtual_chan value=“2” />
      </defaults>
      <router id=“r_x0_y0” >
        <port id=“east” buff_depth=“10”/>
        <port id=“resource” />
        <latency cycles=“3” />
```

```
        <parameter name="proprietary_param" value="hpk" />
      </router>
      <Other routers here...>
    </router_list>

    <link_list>
      <defaults> ... </defaults>
      <link id="l0"> ...</link>
      <link id="l1">
        <src router_ref="r_x0_y0" port_ref="east" />
        <dst router_ref="r_x1_y0" port_ref="west" />
        <pipeline_depth value="1" />
      </link>
      <link id="l2"> ...</link>
    </link_list>

    <terminal_list>
      <defaults> ... </defaults>
      <terminal id="0"> <!-- direct connection on the left of Fig. 6 -->
          <connection link_ref="l0" />
      </terminal>
      <terminal id="1">
          <connection link_ref="l0" />
      </terminal>

      <terminal id="2"> <!-- middle of Fig. 6 -->
          <connection router_ref="r_x0_y0"  port_ref="resource">
      </terminal>
      <terminal id="3">  <!-- right side of Fig. 6 -->
        <connection router_ref="r_x1_y0" port_ref="resource">
        <address value="0x35000" />
        <network_interface type="OCP/AXI/other">
          <delivery value="in-order" drop_pkts="no"/>...
        </network interface>
      </terminal>
      <Other terminals here...>
    </terminal_list>


    <parameter name="param_for_blackbox_noc" value="cmx" />
    <Other arbitrary params that have no special tags in XML />
  </noc>
</platform>

<measurements>
  <simulation_time sec="500e-3" /> <!-- i.e. 500 ms -->
  <cost_function f="A*t_p1" />
  <Other measurement settings>
</measurements>

</system_description>
```