The Environment for Mapping SystemC Multi-module Specifications onto NoC Architectures

Stanisław Deniziak* and Robert Tomaszewski

Department of Computer Engineering, Cracow University of Technology, Cracow, Poland

Department of Computer Science, Kielce University of Technology, Kielce, Poland

Abstract: This work presents a methodology for mapping of a SystemC specification onto a given Network-on-Chip (NoC) architecture, for the purpose of FPGA prototyping. A communication protocol and routing tables are generated automatically using inter-module communication analysis. For each processor in the target architecture, assigned SystemC processes are converted into C++ programs, where all communication method calls are replaced with sending/receiving messages to/from the network interface (NI) process. For each module implemented in hardware a VHDL code of the NI is generated. NIs convert transmitted data into/from network packets according to the communication protocol. The main advantage of our approach is the possibility to prototype and to evaluate many NoC architectures for a given system, without the necessity of modification of the source system specification. Presented embedded HTTP server example substantiates the benefits of the methodology.

Keywords: Network on Chip, SystemC, rapid prototyping, embedded systems.

1. INTRODUCTION

Network-on-Chip (NoC) has been proposed as a new class of architectures for multiprocessor SOC systems [1]. In the NoC, communication is implemented using on-chip packet-switched networks, instead of shared buses, to avoid a communication bottleneck. To become viable, NoC requires support by design tools, especially tools for modeling, design space exploration, evaluation, and synthesis.

One of the best methods of evaluation for digital systems is FPGA prototyping [2]. Physical prototypes enable the estimation of system properties with high accuracy. Modern FPGAs have a complexity about 10 millions gates allowing implementation of multiprocessor systems. Thus, this technology is also used for prototyping of NoC systems [3].

Design process begins by creating a system-level executable specification, so called virtual prototype. Then, virtual prototype is mapped into physical prototype. One of the most widely accepted system-level specification language is SystemC [4]. It supports a powerful generic model of computation, which enables definition of a wide range of different customizable methods of communication and synchronization between processes. SystemC has proven to be suitable for designing of embedded systems, and it is also used for modeling of NoC systems.

In this work a framework for rapid prototyping of NoC systems is presented. It takes an implementation-independent system-level specification given in SystemC and a target NoC topology description as inputs. As the result, C++ programs, routing tables and a RTL VHDL implementation of the NoC are generated. The whole process is fully automated and the output can be implemented as a FPGA prototype, by using standard environment for SOPC (System On a Programmable Chip) implementation. The framework consists of a library of software and hardware components (routers, communication protocols, etc.) and a set of tools for system analysis, software generation and route selection. The main goal of this work is to minimize an effort required for exploration and evaluation of different NoC architectures for a given system, and to provide open methodology, easily extensible. Thus, the unique feature of our approach is that we give the environment to a designer, where one SystemC model can be converted into functioning FPGA prototype, according to different NoC architecture rules. Such prototype may be then a subject of further research leading to the best solution for particular application.

The rest of this paper is organized as follows. The next section reviews related work. Section 3 outlines the main system-level features of SystemC. Overall methodology is explained in section 4. In section 5 an example is presented. Finally, the conclusions are given in section 6.

2. RELATED WORK

A lot of approaches use SystemC for system-level specification of NoCs. In [5] a universal communication API for NoC specification and modeling is presented. Similar SystemC library for asynchronous NoC modeling is presented in [6]. SystemC transaction-level simulation is used in [7] for NoC verification. XpipesCompiler [8] generates SystemC cycle-accurate models for manually specified NoCs. An implementation in SystemC of a scalable and parametrical NoC with a mesh topology is described in [9]. SystemC has been

*Address correspondence to this author at the Department of Computer Engineering, Cracow University of Technology, Cracow, Poland;
E-mail: s.deniziak@computer.org

accepted as a standard-based approach to model SoC/NoC systems [10].

Modern FPGAs offer the possibility of implementation of large systems. Each system can consist of many processor cores and hardware modules, therefore efficient implementation of NoCs in FPGAs is an active area of research. Several NoC based systems implemented in FPGAs are presented in [11-13]. FPGA technology is also used for NoC emulation [14] and to accelerate simulation speed [15], but it is most useful for NoC prototyping [16, 17], enabling architecture exploration, power estimation and accurate functional validation.

Rapid prototyping requires a methodology of mapping system specification onto target architecture in a single flow. Most of the presented design methods are limited to network topology generation. In [18] graph models are used for communication synthesis. A method of mapping system level task graph onto optimal NoC with a mesh topology is presented in [3]. An automatic mapping of DSP algorithms represented as dataflow graphs into a given NoC topology is described in [19]. In [20] a complete design flow, which generates a synthesizable RTL code of a NoC topology, is presented. The integrated environment for synthesis of complete NoC-based SOCs is described in [21]. User configures processing cores, network interfaces, network topology and communication protocols. Then, the target system is generated using configuration files written in XML and component specifications given in Handel-C. The Aethereal [7], NetChip (xpipes [8]) and Pirate [28] design approaches use their own input model formats, and SystemC description is generated as intermediate stage before synthesis. Nostrum [29] focuses only on mesh-based NoC topologies.

According to our best knowledge, there is no method for automatic mapping SystemC specification onto a target NoC-based architecture. In all above approaches SystemC is used for modeling or is a product of design flow. Synthesizable models are given in other languages like VHDL, Handel-C, XML files or by using abstract representations like task graphs. Moreover, in the only complete design flow [21], a functional model of whole system is not used at all. Thus, the functional validation is not available until a physical prototype is implemented. From the other side, using different models for functional validation and for synthesis may cause inconsistency in the system design.

Taking into consideration disadvantages of the existing design environments for NoC-based systems, in this paper, we propose another design flow. In our approach the same specification is used for system-level functional validation and for implementation, and is given by designer. All files required for the system implementation (C++ programs, VHDL files) are generated automatically. We believe that our approach is more appropriate for rapid prototyping, significantly reducing an effort required for the system validation. Because every NoC topology can be useful in FPGAs [27], the methodology presented in this work is capable of mapping the same input specification into different networks. In this way, different target architectures may be evaluated, without the necessity of modification of the source SystemC specification – having one model we can generate many NoC prototypes, e.g. by using different topologies.

3. SYSTEMC FEATURES

SystemC [4] is a C++ library for system and hardware design. System-level specification consists of modules communicating using channels. Channels are connected to module ports. Each module contains at least one process. Processes are activated according to a sensitivity list, defined statically or dynamically. SystemC provides the following system-level features: events, processes, channels and interfaces.

3.1. Events

An event is a low-level primitive which is used to construct different forms of synchronization. Wait/notify model is used. Processes waiting for events are suspended. Process resumes its execution when any event (or set of events) from a specified sensitivity list, defined statically or dynamically, occurs. Static sensitivity list for each process is defined in the module constructor and can not be changed during execution. Dynamic sensitivity list is specified using *wait()* method and temporarily overwrite the static list defined for the same process.

3.2. Processes

Three types of processes can be used: threads, methods and clocked threads. Each thread process has its own thread of execution and can be suspended and resumed. The method process is executed entirely (*wait()* is not allowed) each time after activation. The clocked thread process is a thread process which is only triggered on one edge of one clock.

Processes are created statically or dynamically. Static processes are created during elaboration. Dynamic ones are created during simulation.

3.3. Interfaces

An interface specifies a set of methods to be implemented within channel. Only ports matching given interface type may be used with channels implementing this interface. The main role of interfaces is to separate transmissions from computations, i.e. interfaces provide modules with an access to communication methods implemented in channels.

3.4. Channels

A channel implements one or more interfaces. Primitive and hierarchical channels are distinguished. Primitive channels do not contain processes and can not access other channels. Hierarchical channels are modules which may contain processes and other modules, they may also access other channels. In SystemC the following communication channels are predefined: *sc_signal*, *sc_buffer* and *sc_fifo*. Other standard SystemC channels are used for modeling special behaviors like clocks, buses or controlling access to shared resources, and they will not be considered here.

The *sc_signal* is the simplest communication channel. Transmission is without blocking and buffering, any change of signal state generates an event. The following methods are supported:

- *read(), write()* – for data transmission,
- *event(), value_changed_event()* – for detection of signal changes.

The *sc_signal* channel connects at most one driver (i.e. *sc_out* output port or *sc_inout* bidirectional port) and arbitrary number of *sc_in* input ports.

The *sc_buffer* is a special kind of signal. The only difference is that in case of *sc_buffer* an event is generated after every write, even if it does not change a state.

The *sc_fifo* enables transmissions with buffering and with blocking or without blocking of communicating processes. The following methods are provided:

- *read(), write()* – for transmission with blocking,
- *nb_read(), nb_write()* – for transmission without blocking,
- *num_available(), num_free()* – for checking a state of the FIFO buffer,
- data_written_event(), data_read_event() – for detection of FIFO changes.

The *sc_fifo* channel connects at most one output port (*sc_fifo_out*) with at most one input port (*sc_fifo_in*). Bidirectional ports are not allowed. Read data are removed from the FIFO. Operations without blocking do not change the FIFO state when a read (write) is not possible, i.e. when the buffer is empty (full).

4. RAPID PROTOTYPING METHODOLOGY

Fig. (**1**) shows the rapid prototyping flow diagram. It is assumed that a description of a target NoC topology is specified in XML file. It may be created manually or it is a result of system synthesis. The mapping of the SystemC specification into FPGA prototype with the given NoC topology is performed as follows:

- SystemC specification is analyzed in respect of communication flow, the result is a system communication model described in XML format,
- specifications of modules assigned to processors (IP_S) are translated into C++ programs,
- for each module assigned to hardware component (IP_H) a network interface (NI) is generated,
- communication channels connecting modules assigned to different processors/modules are mapped into the NoC configuration.

As a result, a RTL VHDL specification of the whole system architecture is generated. Physical prototypes are implemented using commercially available tools for target FPGA devices. In our prototyping environment we use Altera's FPGAs with Nios II processor cores, Quartus II system and Nios II Integrated Development Environment [22].

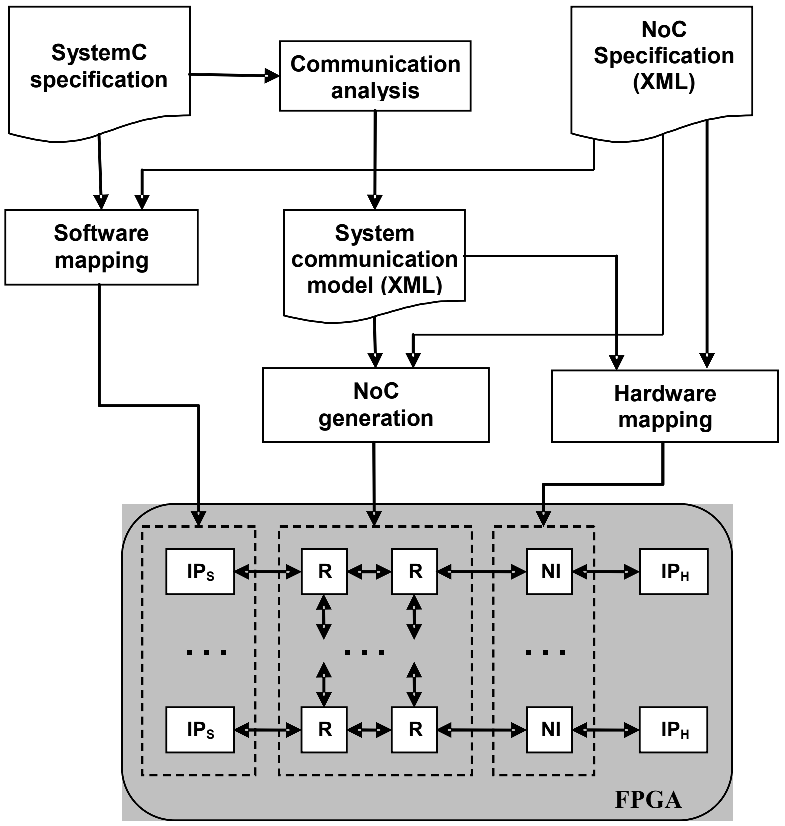


Fig. (1). Rapid prototyping flow.

4.1. Target Architecture Description

The system communication model and target NoC architecture are described in XML format, because of its human readability, robustness and extensibility.

The first XML file we use is *system_sc.xml*, which contains description of SystemC modules with emphasis on communication aspects. The most essential part is what processes are located in SystemC modules, which ports of each module are used by these processes and what channels are used for connecting modules *via* ports. This information is needed for checking the target NoC architecture feasibility, calculation routing paths and packet construction. For example – target system may be not feasible if routers were improperly linked (which results in lack of routing path) or if not all modules were assigned to IPs (Intellectual Property cores).

On Fig. (**2**) we depicted sample architecture described in SystemC and corresponding s*ystem_sc.xml* file. That file is created automatically (Fig. **1**). Every port has its type, list of connected processes, and size of transmission specified in bits. Similarly the channels, but with one exception – *sc_fifo* channels have buffer size defined.

The second file we use is *system_noc.xml*, which consists of description of routers linking (i.e. topology of the network) and mapping of SystemC modules into IP cores. We decided to use LiPaR router [23] which involves using one specialized port (Local Port) for IP linking. The rest of router ports (called, due to their location, North, East, South, East) are used for inter-router connections.

Fig. (**3**) shows scheme of the target NoC architecture and its description in *system_noc.xml* file. For each IP implemented as hardware component (with attribute *impl=”HW”*) widths of all ports, specified in bits, are given. Ports of module which has to be SW implemented don't need *width* attribute at all.

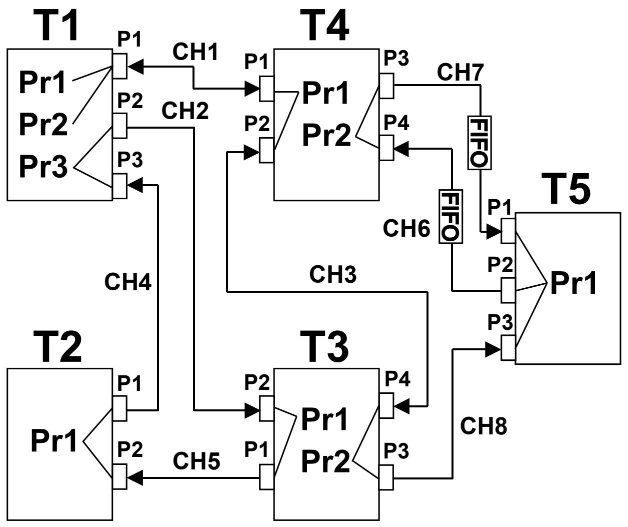
At the current stage of our research we focused only on standard types of ports and channels offered by SystemC, but using XML for description lets us easily extend presented methodology and include other types of ports and channels (e.g. OCCN channels [5]). The only thing we should know is how they work, to implement it in our mapping tool.

4.2. Mapping SystemC Modules into Software

Embedded software is generated by replacing SystemC simulation kernel with real time operating system (RTOS) calls [24]. Communication channels connecting modules assigned to different IPs are replaced with the corresponding channels (*signal2ni*, *buffer2ni*, *fifo2ni*) redirecting transmissions to/from the NI. In the programming environment for a target processor the SystemC library is replaced with the corresponding implementation library. In this way, the SystemC source code may be compiled in a RTOS environment without any modification.

Channels *xxx2ni* implement the same interfaces that related SystemC channels, but instead of communicating with destination processes they send/receive data to/from a

(**a**)



(**b**)

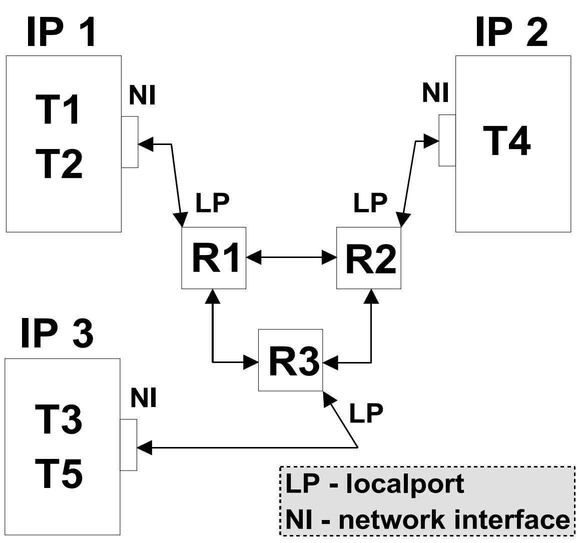
```
<arch id=”sample_SC_chip”>
 <module id=”T1”>
  <process id=”PR1”>1</process>
  <process id=”PR2”>2</process>
  <process id=”PR3”>3</process>
  <port id=”P1” type=”sc_inout”
      process_conn=”1 2” size=”16”>1</port>
  <port id=”P2” type=”sc_out”
      process_conn=”3” size=”64”>2</port>
  <port id=”P3” type=”sc_in”
      process_conn=”3” size=”32”>3</port>
 </module>
 <module id=”T4”>
  <process id=”PR1”>1</process>
  <process id=”PR2”>2</process>
  <port id=”P1” type=”sc_inout”
      process_conn=”1” size=”16”>1</port>
  <port id=”P2” type=”sc_inout”
      process_conn=”1” size=”8”>2</port>
  <port id=”P3” type=”sc_fifo_out”
      process_conn=”2” size=”32”>3</port>
  <port id=”P4” type=”sc_fifo_in”
      process_conn=”2” size=”64”>4</port>
 </module>
...
 <channel id=”CH1” type=”sc_signal”
      conn=”T1_P1 T4_P1”>1</channel>
 <channel id=”CH6” type=”sc_fifo”
      conn=”T4_P4    T5_P2”
      buffer=”5”>6</channel>
 <channel id=”CH7” type=”sc_fifo”
      conn=”T4_P3    T5_P1”
      buffer=”5”>7</channel>
</arch>
```

Fig. (2). Sample SystemC architecture scheme (**a**) and part of corresponding system_sc.xml (**b**).

process implementing NI. Local module and port addresses are also stored in these channels. NI communicates directly with the local port of the router using low-level I/O operations supported by RTOS. Other details about NI functionality are given in the next subsections.

The mapping of a SystemC specification into embedded software is presented on Fig. (**4**). Assume that modules M1 and M2 are assigned to one Nios II processor. Thus, the specification of these modules is copied to the Nios II programming environment, without any modification. Next, the *main()* routine (*Proc1.cpp*) instantiating processes and communication channels is generated. We apply MicroC/OSII

(**a**)



(**b**)

```
<noc id=”sample_NoC_chip”>
 <router id=”R1” l_port=”IP1” E_port=”R2”
     S_port=”R3” type=”lipar”>1</router>
 <router id=”R2” l_port=”IP2” W_port=”R1”
     S_port=”R3” type=”lipar”>2</router>
 <router id=”R3” l_port=”IP3” E_port=”R2”
     W_port=”R1” type=”lipar”>3</router>
 <IP id=”IP1” impl=”HW” sc_mod=”T1 T2”>1
  <port id=”P1” width=”8” />
  <port id=”P2” width=”1” />
  <port id=”P3” width=”8” />
  <port id=”P4” width=”16” />
  <port id=”P5” width=”8” />
 </IP>
 <IP id=”IP2” impl=”SW” sc_mod=”T4”>2</IP>
 <IP id=”IP3” impl=”SW” sc_mod=”T3
     T5”>3</IP>
</noc>
```

Fig. (3). Sample target NoC architecture scheme (**a**) and corresponding system_noc.xml (**b**).

RTOS, which fully supports concurrent execution of tasks, so the process semantics of SystemC is respected.

4.3. Mapping SystemC Modules Onto Hardware IPs

We assume that hardware modules are available as IP components or they are synthesized using commercially available tools. Thus, in our environment only a VHDL code, instantiating these components, is generated.

Each IP$_H$ communicates with NoC through dedicated NI (see Fig. **5**). The module ID (index) and IDs of ports are stored in the NI. Since the functionality of the NI depends on type of module ports, different VHDL code is generated for each NI.

Communication is done by use of two handshake signals (*Req/Ack*) between the co-operating ports (output and input) - similarly to LiPaR [23] synchronization of transmission. If designer uses custom IP$_H$ module, supporting different handshaking/transmitting protocol, he's responsible for providing a converting adapter, adjusting IP interface to our NI. If *size* and *width* attributes, specified for the same port in *system_sc.xml* and *system.noc.xml* files, are not equal, then single transmission between IP and NI is *width*-length and all data are transmitted in blocks (the number of blocks is *(size + (width - 1)) div width*). Otherwise one *size*-length transmission is needed.

4.4. Mapping Channels into Communication Network

As we mentioned before the router of our choice is LiPaR, because of it's advantages in FPGA implementations [3, 23]. By default this router use XY routing algorithm, which is best suited for mesh based topologies. In our methodology we cannot predict the type of the target topology. Generally – it can be regular or irregular topology, so XY routing (based on Cartesian co-ordinates) should be replaced by more universal routing algorithm. We chose deterministic

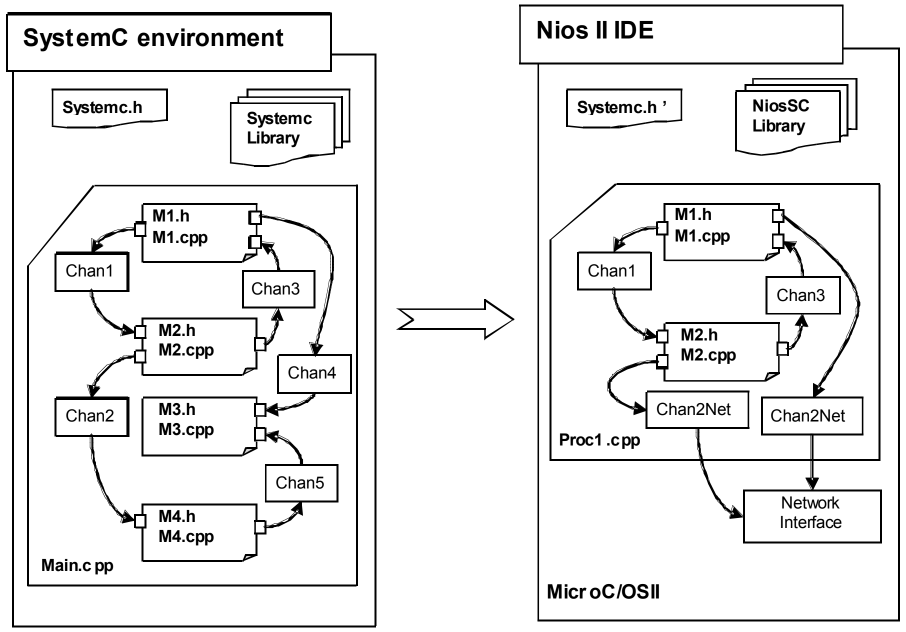


Fig. (4). Embedded software generation.

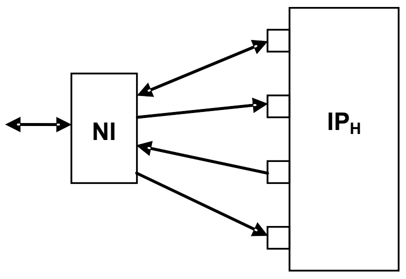


Fig. (5). Mapping of a hardware component.

shortest-path routing, which involves slight modification of decoding logic of the router. The Crosspoint Matrix is driven not by FSM Controller, but by content of memory block, holding static routing table. The address of destination IP (taken form header of the packet) sets the row in memory and the addressed row contains control information for Crosspoint Matrix (i.e. which output channel to use – North, East, South, West or local).

Because all ports of the routers are bidirectional with inner FIFOs, every type of SystemC channel (*sc_signal, sc_buffer, sc_fifo*) is treated by routers in the same way – as path from source IP to destination IP. Write operation is treated as sending the packet, read – as receiving. The exact type of channel matters for NIs and is described further.

For building the routing information we use BSF based algorithm [25]. It is the graph algorithm for searching the shortest paths between one source vertex and the rest of vertices. In the first step (Fig. **6**) network topology of NoC is mapped onto graph – IPs and routers are vertices, links betweens vertices are edges of the graph. Edges are without the weight and direction. Nodes have information about type (IP or router) and connection, i.e. which port is connected to particular edge to next router/IP (essential for routers). That mapping is done with use of *system_noc.xml* file. Next, for every channel described in *system_sc.xml* file we determine IP modules used in channel connection. For that IPs we apply BSF algorithm. The result is stored in the list of predecessors and successors (i.e. routers) on the path between communicating IP cores (precisely – their NIs). If the list would be empty – that means, that target architecture is not feasible and needs redesign. Every position on the list contains the number/identifier of next-hop router and name/identifier of the port used as link (edge in the graph). Due to the form of returned information by BSF (reverse order of vertices on the path) for IN and OUT ports we treat IN as a source vertex and OUT as a destination. For INOUT port it's meaningless.

The packet format (Fig. **8**) is simple and standard LiPaR format compatible [23] with one exception – instead of XY coordinates, header contains destination IP identifier (address). That identifier is used by every router to address routing memory (address decoding logic) and decides, which port will be involved in further transmission.

The last important element is NI. It is responsible for forming packets and sending/receiving (write/read) operations. Forming packet includes address translation from local format to global. Local format (i.e. between IP core processes and NI) defines port ID (as described in *system_sc.xml* file), global designates target IP.

```
GenerateRoutes()
 BuildGraph(in: system_noc.xml,out: Graph);
 BuildListOfChannels(in: system_sc.xml,
 while ListOfChannels <> nil do
    Channel := ListOfChannels.Current;
    foreach Channel.Connection do
      if Channel.Connection.Type = IN or
                BSF(in: Graph,
                    out: ListOfPaths);
    if any ListOfPaths = nil then return
    Add(inout: CompleteListOfPaths,
        in: ListOfPaths)
    ListOfChannels := ListOfChannels.Next;
 end;
 BuildLocalRoutingTables();
end;
```

Fig. (6). Algorithm for shortest-path generation.

If the algorithm shown on the Fig. (**6**) won't raise any error we can program routing memories in routers (Fig. **7**).

```
BuildLocalRoutingTables()
 BuildListOfRouters(in: system_noc.xml,
           out: ListOfRouters)
 while ListOfRouters <> nil do
    Router := ListOfRouters.Current;
    while CompleteListOfPaths <> nil do
       Path := CompleteListOfPaths.Current;
       destIP := Path.Head;
       while Path <> nil do
          node := Path.Current;
          if node == Router then
            Router.table[destIP] :=
               portID_to_next_on path
           Path.Current := Path.Next;
       end;
       CompleteListOfPaths.Current :=
             CompleteListOfPaths.Next;
    ListOfRouters.Current :=
                 ListOfRouters.Next;
end;
```

Fig. (7). Building local routing tables.

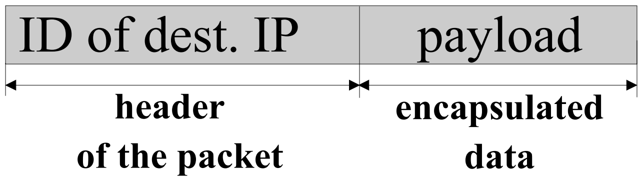


Fig. (8). Structure of the frame sent between routers and NIs.

Dest IP	Local Addr	Dest Addr	Multi

Fig. (9). NI's translating table scheme.

Every NI has a table filled with portID to IP's ID mappings (see Fig. **9**). These tables are created according to information placed in *system_sc.xml* and *system_noc.xml* files. To make sure, that IDs of the local ports are unique, they are preceded by their SC module ID (see *system_sc.xml* file). There are two tables – one for resolving remote addresses (sending/writing) and one for local (receiving/reading).

The fields: *Local Sender Addr*, *Local Dest Addr, Length of Data, Number* and optionally technical part (e.g. to inform about blocking/non-blocking operation) form a header of NI (transport layer), which is entirely encapsulated with data as a payload in packets for routers (network layer). *Local Sender Addr* and *Number* fields are needed by acknowledgment mechanism for FIFO channels described later. In case of the transmission is multicast type (e.g. one OUT port is connected through channel to many IN ports in SystemC model), the table has address of position (field *Multi*), where next receiver is described. Ordinary transmissions and last receiver in multicast case have 0 in this field. Multicast transmissions demand multiple frames sending.

Every local port of NI has its own input/output address and buffer capable to store maximum process write/read data package (see *size* attribute for *port* tag in *system_sc.xml*). Writing to port means sending data to a particular interface (buffer) of NI, reading – getting content of the buffer.

When inner process requests write through port – it sends request and data to appropriate interface on NI. Request has a form of a packet with header described earlier. Having local address of destination, NI seeks for destination IP address and forms the frame. All packet data are treated as a payload for a frame. Header of a frame we described previously (see Fig. **8**). It's determined by router construction.

If a size of a packet exceeds the MTU (Maximum Transport Unit – allowed size of a frame) of routers (determined by their FIFOs) the packet is sent in multiple pieces. In case of deterministic routing every frame is traveling the same route and in constant order (unlike the classic computer IP-based networks) so assembling by receiver doesn't need any additional information about frame/packet number (i.e. every properly addressed part of a packet is written into receiver's FIFO without the header). In contrast, adaptive routing protocols (not considered here) demand other construction of the router, and allow for different paths from source to destination. Such approach causes, that frames can arrive with different delays and need some kind of information for arrangement. It is achieved by offset number for a frame, put after *ID of destination IP* field in frame (compare with Fig. **8**).

Receiver NI (after removing header of a frame) decodes packet header, seeks for proper interface (i.e. port for IP core) and stores it. To limit the traffic in the NoC we decided to implement buffers on both, sender and receiver, sides. To ensure, for *sc_fifo* channels, buffers integrity we introduced the mechanism of acknowledgment – each FIFO transmission causes sending back of small, technical frame with number of packet being acknowledged. Then, next write operation can be performed.

If the header contains information about blocking mode of transmission, the NI uses MicroC/OSII low-level mechanisms to stop/resume involved process. Also SystemC channel operations for event/buffer state checking are implemented through MicroC/OSII inter process communication mechanisms. For example if the NI has a blocking read (write) request and the addressed FIFO buffer is empty (full) it sends the local signal to the local process forcing it to wait.

Channel mapping details:

- *sc_signal* - local buffers for NIs are one-element, write operation transmits data only when current data is different from previously written in buffer, no acknowledgment is needed;
- *sc_buffer* – like for sc_signal, but write operations always force the transmission, no acknowledgment is needed;
- *sc_fifo* – local buffers of transmitter and receiver have as many elements as in specification (see *buffer* attribute for *port* tag in *system_sc.xml*). Two copies of the same buffer exist in receiver and transmitter NI. Only write operation generates transmission. Acknowledgments are needed.

5. EXAMPLE

For the purpose of demonstration of our methodology we implemented in SystemC embedded WWW server and mapped it onto NoC architecture. Sample server supports selected commands of the HTTP protocol [26] (i.e. GET and POST). HTTP protocol is well-known mechanism used in Internet for serving WWW resources (HTML pages with embedded elements, e.g. graphics). The client (WWW browser like Internet Explorer, Firefox, Opera, etc.) sends a request to server in form of GET/POST message. The request is initiated after confirming the WWW address put into address bar of the browser and consists of several lines of text. These, so called, HTTP headers describe capabilities of a client and give precise information about requested resource (file name, preferred language of document, transmission mode, time stamp, etc.). Then server searches its local file system, retrieves demanded resource and sends it back to client including response code (success, temporary error, permanent failure). HTTP is an application protocol, which uses TCP/IP as transport/network protocols. The fabric of the server is depicted on Fig. (**10**) and consists of modules:

- *GetReq* – includes one process waiting for requests on port 80. Supports max. 6 simultaneous connections. Every connection has it's own, separate port for transmitting data. All requests are sent to the *ProcReq* module. The *Transmit* module is informed after every flushing of the buffer of the transmitter and send next fragments of the requested files. The *GetReq* supports requests in the TCP/IP style – by calling functions socket, bind, listen and accept.
- *ProcReq* – includes one process activated by transmission from *GetReq*. Next, the data is read by *TCP/IP* receive command into the buffer. If there is entire HTTP request, it is analyzed and module *ProcGet* is activated (in case of GET command) or *ProcPost* (in case of POST command).
- *ProcGet* – processes the GET command and sends error message (if file doesn't exist) or header of the found file. *ProcGet* activates *FileSys* – file manager module.
- *ProcPost* – processes the POST command. The called procedure depends on passed parameters.

- *Transmit* – includes one process for transmitting data (i.e. file). The file requested by GET command is received from *FileSys* module in 8kB parts. One execution of the process means sending one part of the file.
- *ManConn* – this process determines the status of the connection. In case of exceeding time limit or raising error during command processing, the connection is closed. Connection can be also closed after normal transmission.

The *TCP/IP* and *FileSys* modules are elements of Altera's NiosII IDE system [22]. The first one implements functions equivalent to standard BSD socket functions. The second one is ZIP file system manager for flash memories. All transmissions use *sc_fifo* channels with one-element buffers, except some buffers of the channels: *gr_to_pr*, *gr_to_tr* and *gr_to_mc*, which have 6 elements buffers.

According to SystemC description depicted as scheme on Fig. (**10**) we obtain **HTTP_sc.xml** file, partially shown on Fig. (**11**) (it contains description for *ProcGet* and *FileSys* modules and their ports connected through *fopen_in* and *fopen_out* channels).

Target NoC configuration and its description *HTTP_noc.xml* file are shown on Fig. (**12**). Note dotted lines defining shortest paths between several IP cores and implemented into them SystemC modules. These sample paths are result of algorithm from Fig. (**6**).

On Fig. (**13**) we present denotation of router ports and a part of routing table (for dotted paths on Fig. **12a**). Having binary coded router ports we are able to program local routing tables inside the routers. That operation needs data shown on Table **1**, which are the result of algorithm from Fig. (**6**).

Table **2** shows structure of translational table for sending purpose implemented inside the NI. SystemC module ID and SystemC port ID form local destination address (determined by channel description in *HTTP_sc.xml* file) translated on corresponding destination IP ID (index). Local source address designates the proper row in table.

6. CONCLUSIONS

This paper presents the methodology for rapid mapping of SystemC model into working FPGA prototype. The architecture of the prototype is NoC based. The exact target NoC architecture is described by designer and thus he can quickly estimate the most suitable architecture for particular specification of the system. Although entire process is complex, we proved it is possible to do it automatically. We defined set of rules and algorithms to successfully perform mapping procedure. The main stress was laid on proper implementation of the communication aspects, i.e. translation channels semantics on NoCs communication. Example of HTTP server illustrates our methodology.

Most of the works about NoC architectures is focused on particular aspects of NoC modeling and optimization. In our work we enclosed hitherto existing achievements in one methodology, which lets designer not only model, but also to obtain real, feasible FPGA prototype. Thus, it gives huge degree of flexibility in making decision about the most

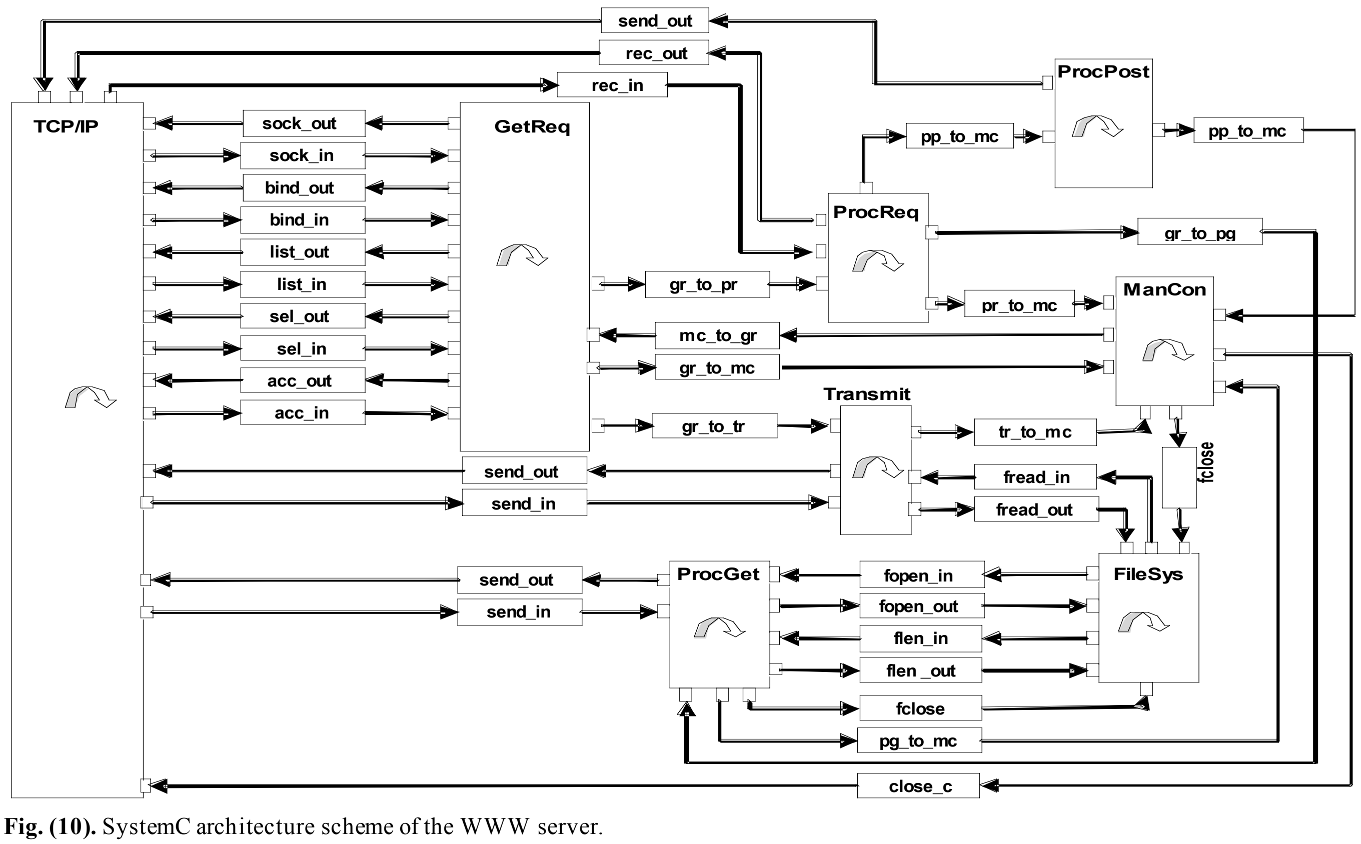


Fig. (10). SystemC architecture scheme of the WWW server.

```
<arch id="HTTP_SC">
<module id="010"> <!-- ProcGet module -->
 <process id="0">0</process>
 <!-- ports connected to fopen_xx channels -->
 <port id="0000" type="sc_fifo_in"
       process_conn="0" size="32">0</port>
  <port id="0001" type="sc_fifo_out"
       process_conn="0" size="2072">1</port>
  ...
 </module>
<module id="001"> <!-- FileSys module -->
  <process id="0">0</process>
 <!-- ports connected to fopen_xx channels -->
  <port id="0000" type="sc_fifo_out"
       process_conn="0" size="32">0</port>
 <port id="0001" type="sc_fifo_in"
       process_conn="0" size="2072">1</port>
  ...
</module>
 <!-- fopen_in channel -->
<channel id="000000" type="sc_fifo" buffer="1"
conn="001_0000 010_0000">0</channel>
<!-- fopen_out channel -->
<channel id="000001" type="sc_fifo" buffer="1"
conn="001_0001 010_0001">1</channel>
</arch>
```

Fig. (11). Part of HTTP_sc.xml.suitable target NoC configuration for specific SystemC description.

Although of many advantages we have to remember, that automatic mapping methodology has also some unfavorable impact on cost factor of target architecture. Is is due to SystemC channel semantics compatibility. To maintain functionality of SystemC channel in conformity with NoC assumptions we had to agree on some excess in communication layer. It includes acknowledgment mechanism, translating addresses format (with use of translational tables in NIs), additional sending/ receiving buffers, messages fragmentation/ defragmentation procedures, etc.. Such inconveniences are also present in other common NoC modeling/prototyping approaches and are acceptable.

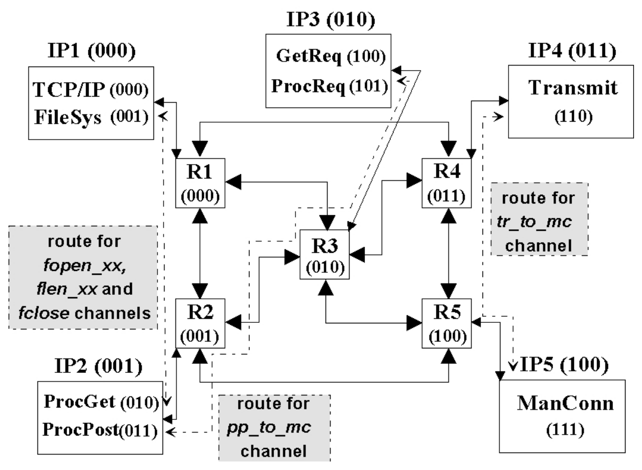
Below we enumerate key features/capabilities of our approach in contrast to the other methodologies:

- one SystemC input description (only Nostrum uses SystemC as input, other approaches use their own, manually configurable, input description formats like XML files)
- any NoC topology (others are often limited to mesh-based topology, e.g. Nostrum)
- any router/routing strategy (competitors use mainly deterministic routing or partially adaptive, where adaptivity is incorporated into routers increasing NoC complexity)
- modeling/prototyping aware (model is given as input in SystemC, prototype is generated in VHDL; some of other frameworks are used only for simulation purposes, e.g. Nostrum, Noxim)

In this paper we focused on selected SystemC channels, but it is worth to notice, that methodology is open and easy extensible – mapping is not limited only to *sc_signal channel, sc_fifo channel,* LiPaR routers, *etc.*

We gained an acceptation for our idea [30] and we are still working under improvement of our methodology. Simultaneously we applied it successfully for development of new routing strategies [31].

(**a**)



(**b**)

```
<noc id="HTTP_NoC">
 <router id="000" l_port="000" N_port="011"
       E_port="010" S_port="001"
       type="lipar">0</router>
 <router id="001" l_port="001" N_port="000"
       E_port="010" S_port="100"
       type="lipar">1</router>
 <router id="010" l_port="010" N_port="000"
       E_port="011" S_port="100" W_port="001"
       type="lipar">2</router>
 <router id="011" l_port="011" N_port="000"
       S_port="100" W_port="010"
       type="lipar">3</router>
 <router id="100" l_port="100" N_port="011"
       S_port="001" E_port="010"
       type="lipar">4</router>
 <IP id="000" impl="SW" sc_mod="000 001">0
       </IP>
 <IP id="001" impl="SW" sc_mod="010 011">1
       </IP>
 <IP id="010" impl="SW" sc_mod="100 101">2
       </IP>
 <IP id="011" impl="SW" sc_mod="110">3</IP>
 <IP id="100" impl="HW" sc_mod="111">4
   <port id="cin0" width="8" />
   <port id="cin1" width="8" />
   <port id="cin2" width="8" />
   <port id="cin3" width="8" />
   <port id="cin4" width="8" />
   <port id="cout" width="8" />
   <port id="fclose" width="32" />
   <port id="cclose" width="16" />
 </IP>
</noc>
```

Fig. (12). Scheme of target NoC architecture with binary IDs (**a**) and corresponding HTTP_noc.xml (**b**).

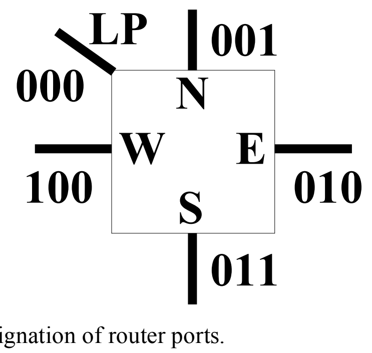


Fig. (13). Designation of router ports.

Table 1. Part of Routing Table

Src IP	R1	R2	R3	R4	R5	Dest IP
000	011	000	-	-	-	001
001	000	001	-	-	-	000
001	-	010	000	-	-	010
010	-	000	100	-	-	001
011	-	-	-	011	000	100
100	-	-	-	000	001	011

Table 2. Part of Translational Table of NI for Purpose of Forming a Header of a Frame Before Sending (*Multi* Field Omitted)

Dest SC_module|SC_port	Dest IP	Comment
010|0001	001	FileSys sends through *fopen_in* channel
001|0001	000	ProcGet sends through *fopen_out* channel

Next step in our future work will be implementing other types of routers, different routing strategies and support for custom SystemC channels.

Table **3** demonstrates sample local routing table for router R2. The content of routing memory (essential part of routing logic) is built according to information gained in the table depicted in Table **1**. *Dest addres* is used for addressing memory and the content of addressed row specifies proper *output port* for further transmission.

Table 3. Sample Part of Local Routing Table for Router R2 (Compare with Table 1)

Dest. Address	Output Port
000	001
001	000
010	010
...	...

REFERENCES

[1] L. Benini and G. De Michelli, “Networks on Chips: A New SOC Paradigm”, *IEEE Comput.,* pp. 70-78, January 2002.

[2] D. Kissler, A. Kupriyanov, F. Hannig, D. Koch, and J. Teich, “A Generic Framework for Rapid Prototyping of System-on-Chip Designs”, *Proc. of the CDES*, pp. 189-195, 2006.

[3] B. Sethuraman and R. Vemuri, “optiMap: A Tool for Automated Generation of NoC Architectures using Multi-Port Routers for FPGAs”, *Proc. of the DATE*, pp. 947-952, 2006.

[4] IEEE Standard SystemC Language Reference Manual, IEEE, New York, 2006.

[5] M. Coppola, S. Curaba, M. D. Grammatikakis, G. Maruccia, and F. Papariello, “OCCN: A Network-On-Chip Modeling and Simulation Framework”, *Proc. of the DATE Designers’ Forum*, pp. 174-179, 2004.

[6] C. Koch-Hofer, M. Renaudin, Y. Thonnart, and P. Vivet, “ASC, a SystemC extension for Modeling Asynchronous Systems, and its application to an Asynchronous NoC”, *Proc. of the NOCS*, pp. 295-306, 2007.

[7] K. Goossens, J. Dielissen, O. P. Gangwal, S. G. Pestana, A. Radulescu, and E. Rijpkema, “A Design Flow for Application-Specific Networks on Chip with Guaranteed Performance to Accelerate SOC Design and Verification”, *Proc. of the DATE*, pp. 1182-1187, 2005.

[8] A. Jalabert, S. Murali, L. Benini, and G. de Michelli, “XpipesCompiler: A Tool for Instantiating Application Specific Networks on Chip”, *Proc. of the DATE*, pp. 884-889, 2004.

[9] A. Portero, R. Pla, and J. Carrabina, “SystemC implementation of a NoC”, *Proc. of the IEEE Int. Conf. Industrial Technology*, pp. 1132-1135, 2005.

[10] T. Kogel, R. Leupers, and H. Meyr, “Integrated System-Level Modeling of Network-on-Chip enabled Multi-Processor Platforms”, Springer, 2006.

[11] T. A. Bartic *et al.*, “Highly Scalable Network on Chip for Reconfigurable Systems”, *Proc. of the International Conference on System-On-Chip*, pp. 79-82, 2003.

[12] S. Tota, M. Casu, P. Motto, M. R. Roch, and M. Zamboni, “NoCRay, an FPGA Network-on-Chip Based MP-SoC for Graphics Ray Tracing Applications”, *Proc. of DATE*, 2007.

[13] M. Schoeberl, “A Time Triggered Network on Chip”, *Proc. of the FPL*, pp. 377-382, 2007.

[14] N. Genko, D. Atienza, G. De Micheli, J. M. Mendias, R. Hermida, and F. Catthoor, “A Complete Network-On-Chip Emulation Framework”, *Proc. of the DATE*, pp. 246-251, 2005.

[15] P. T. Wolkotte, P. K. F. Holzenspies, and G. J. M. Smit, “Fast, Accurate and Detailed NoC Simulations”, *Proc. of the NOCS*, pp. 323-332, 2007.

[16] C. Puttmann, J. C. Niemann, M. Porrmann, and U. Ruckert, “GigaNoC - A Hierarchical Network-on-Chip for Scalable Chip-Multiprocessors”, *Proc. of the Euromicro DSD*, pp. 495-502, 2007.

[17] E. Salminen, A. Kulmala, and T. D. Hamalainen, “HIBI-based multiprocessor SoC on FPGA”, *Proc. of the ISCAS*, vol. 4, pp. 3351-3354, 2005.

[18] A. Pinto, L. P. Carloni, and A. L. Sangiovanni-Vincentelli, “Efficient Synthesis of Networks on Chip”, *Proc. of the ICCD*, 2003.

[19] X. Wu, T. Ragheb, A. Aziz, and Y. Massoud, “Implementing DSP Algorithms with On-Chip Networks”, *Proc. of the NOCS*, pp. 307-316, 2007.

[20] L. Benini, “Application Specific NoC Design”, *Proc. of the DATE*, pp. 491-495, 2006.

[21] A. Kumar, A. Hansson, J. Huisken, and H. Corporaa, “An FPGA Design Flow for Reconfigurable Network-Based Multi-Processor Systems on Chip”, *Proc of the DATE*, pp. 117-122, 2007.

[22] www.altera.com

[23] B. Sethuraman, P. Bhattacharya, J. Khan, and R. Vemuri, “LiPaR: A Light-Weight Parallel Router for FPGA-based Networks-on-Chip”, 15th Great Lakes Symposium on VLSI (GLSVLSI'05), pp. 452-457, 2005.

[24] F. Herrera, H. Posadas, P. Sanches, and E. Villar, “Systematic Embedded Software Generation form SystemC”, *Proc. of the DATE*, pp. 142-147, 2003.

[25] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to algorithms, The MIT Press, 2001.

[26] R. Fielding, J. Gettys, J. Mogul, *et al.*, “Hypertext Transfer Protocol - HTTP/1.1”, RFC 2616.

[27] M. Saldaña, L. Shannon, and P. Chow, "The Routability of Multiprocessor Network Topologies in FPGAs", *Proc. of the 2006 international workshop on System-level interconnect prediction*, pp. 49-56, 2006.

[28] G. Palermo and C. Silvano, “Pirate: A framework for power/performance exploration of network-on-chip architectures”, *Proc. of 14th Int. Workshop on Integrated Circuit and System Design, Power and Timing Modeling, Optimization and Simulation (PTMOS),* 2004.

[29] Z. Lu, I. Sander, and A. Jantsch, “Refinement of a Perfectly Synchronous Communication Model onto Nostrum NoC Best-effort Communication”, *Proc. of the Forum on Design Languages,* 2005.

[30] S. Deniziak and R. Tomaszewski, "Rapid Prototyping of NoC Architectures from a SystemC Specification", accepted for publication in Proceedings of the IEEE Workshop on Design and Diagnostics of Electronic Systems, 2008.

[31] S. Deniziak and R. Tomaszewski, "Adaptive Routing Protocols Validation in NoC Systems *via* Rapid Prototyping", accepted for publication in Proceedings of the IEEE Human System Interaction, 2008.

Received: March 31, 2008 Revised: April 16, 2008 Accepted: April 19, 2008