A highly efficient behavioral model of router for network-on-chip with link aggregation

Korotkyi Ievgen* and Lysenko Oleksandr

Department of Design of Electronic Digital Equipment (DEDEC),
National Technical University of Ukraine "Kyiv Polytechnic Institute",
Polytechnic 16 street, 03056 Kyiv, Ukraine
E-mail: korotkiy.eugene@ieee.org o.lysenko@kpi.ua
*Corresponding author

Abstract: A new method of link aggregation (LAG) in networks-on-chip (NoC) is investigated. This method considerably (by 152÷300%) increases a network saturation threshold by connection of topologically adjacent routers using of multiple physical links. The authors consider several ways how to improve simulation performance of digital circuits. An algorithm, which illustrates operation of high-performance SystemC model of a NoC router with LAG, is proposed. The paper also includes results of computer simulation of the suggested fast SystemC model and its synthesizable System Verilog counterpart. The results of ModelSim simulation show that usage of proposed SystemC model instead of its System Verilog analog reduces the duration of simulation in 10 times and diminish the required memory in 121 times. Herewith the prediction error for saturation threshold of NoC does not exceed 6.1%.

Keywords: network-on-chip; NoC; link aggregation; LAG; router; FPGA; simulation; SystemC.

Reference to this paper should be made as follows: Korotkyi, I. and Lysenko, O. (2012) 'A highly efficient behavioural model of router for network-on-chip with link aggregation', *Int. J. Embedded Systems*, Vol.x, No.x, pp. x–x.

Biographical notes: Korotkyi Ievgen is a PhD candidate in the Department of Design of Electronic Digital Equipment (Faculty of Electronics) at the National Technical University of Ukraine "Kyiv Polytechnic Institute". His research focuses on digital design, networks-on-chip, systems-on-chip, embedded systems and processor design. He received his Bachelors Degree and Masters Degree in Electronic Devices from the National Technical University of Ukraine "Kyiv Polytechnic Institute" in 2007 and 2009.

Oleksandr Lysenko is a Professor and a Head of Department of Design of Electronic Digital Equipment in the Faculty of Electronics at the National Technical University of Ukraine "Kyiv Polytechnic Institute". His research focuses on methods and means of digital processing of audio & video signals, including medical applications, SoC and NoC design. He received a Dr. Sc. Techn. Degree in Biological & Medical Devices & System, a PhD Degree in Devices and Methods for Measuring Mechanical Quantities and a Diploma of Radio-Engineer from the National Technical University of Ukraine "Kyiv Polytechnic Institute".

1 Introduction

One typical feature of systems-on-chip (SoC) nowadays is continuous increase of computational cores, and also higher requirements to the number and capacity of intermodular connections (Bjerregaard and Mahadevan, 2006). The former ways to organize the communication subsystem, such as "common bus" or "fully linked architecture", lose their effectiveness due to complexity of SoC layout (Angiolini, Meloni and Benini, 2007; Marculescu et al., 2007). To solve the problem of data exchange within the VLSI circuits, a concept of network-on-chip (NoC) was suggested (Dally and Towles, 2001). This approach has all the advantages of scalability, parallelism and high clock frequency. An overview of the NoCs is described in papers (Atienza et al., 2008; Marculescu and Bogdan, 2009; Marculescu et al., 2009; Gu, 2011).

The most important performance metrics of NoCs are latency and throughput. Latency it is the time, which elapsed from the moment of packet creation until the moment of it receiving at the destination node, including the queuing time at the source. Throughput is the maximum traffic accepted by the network, where traffic accepted is the amount of information injected into the network per time unit. Throughput is limited by the saturation threshold – the value of accepted traffic when NoC becomes congested and latency increases in tenth of times.

Due to short input queues in NoC routers, it looks unreasonable to buffer a whole packet before sending it to the destination port. Instead, the wormhole flow control is widely used, when the packet is partitioned into "atomic" flow control units (flits), transferred continuously one after another (Dally, 1990). The flits are advancing as far as possible, without waiting for arrival of the followers. It gives lower requirements to the buffer space volume, but the probability of head of line blocking (HOLB) increases dramatically. HOLB arises when two (or more) packets from different input queues require the same output port of the router. In this case, only one request for connection will be satisfied. The input ports, which loose the arbitration, must wait until the required output becomes free, causing HOLB in upstream routers. Since in wormhole flow control one packet can span several routers at the same time, this increases probability of HOLB. As shown in (Dally, 1992), HOLB may reduce NoC throughput to 50% of the network capacity.

One efficient method, which can reduce the HOLB probability is using of virtual channels (VC). However, this approach cannot resolve the HOLB problem completely, since in this case the virtual flows are multiplexed via a single physical connection between neighbouring routers, which imposes a limitation on network saturation threshold value (Korotkyi and Lysenko, 2011).

In (Korotkyi and Lysenko, 2011) we suggested an architectural and hardware solution for implementing link aggregation (LAG) in NoC, which allows to eliminate HOLB and significantly (up to 300%) raise the network saturation threshold value in comparison with classical wormhole NoC. This result becomes possible due to using of several physical links for connection of topologically adjacent routers. One disadvantage of LAG is that it requires large volume of hardware resources for the NoC implementation in comparison with the classical wormhole architecture (without VC). The hardware costs of the NoC with LAG can be reduced by performing optimization the number of physical links inside the aggregated logical connections. To make this optimization, we need a high-performance behavioral model of a NoC with LAG. This is a goal of the present paper.

In this paper we show how to increase the simulation performance of SystemC models and analyze the advances of this language for parametric optimization of integrated circuits (ICs). We also investigate the algorithm of the behavioral SystemC model of a NoC router with LAG. In comparison with the synthesizable System Verilog model of such system, described in (Korotkyi and Lysenko, 2011), our new solution allows to decrease simulation time in 10 times, and diminish the used memory volume in 121 times.

The paper has the following structure. In section 2 we analyze advantages and disadvantages of wormhole NoCs with VC. Section 3 is devoted to the structure and principle of operation of a NoC router with LAG. In section 4 we discuss several ways for improving efficiency of simulation of IC models, and validate the application of SystemC language for this purpose. Next section describes the algorithm of high-performance SystemC model of a NoC router with LAG. In section 6 we use computer simulation to analyze the dependence of NoC saturation threshold on the number of links in aggregated logical channels for two models – a high-performance SystemC one and its System Verilog synthesizable analog. Here we also present a comparative analysis of accuracy and operation efficiency for both models. The last section contains conclusion and outline of further research.

2 Advantages and disadvantages of wormhole NoCs with VC

In previous section we mentioned about HOLB problem in wormhole NoCs. To reduce the probability of HOLB, William Dally (Dally, 1992) proposed architecture of a router with VC. Each input port of such a router contains several queues (VCs), connected to the single input of the port through demultiplexer. In case of blocking of particular VC, the other free VCs can be used for data transmission. Therefore, the probability of HOLB decreases and the network saturation threshold increases.

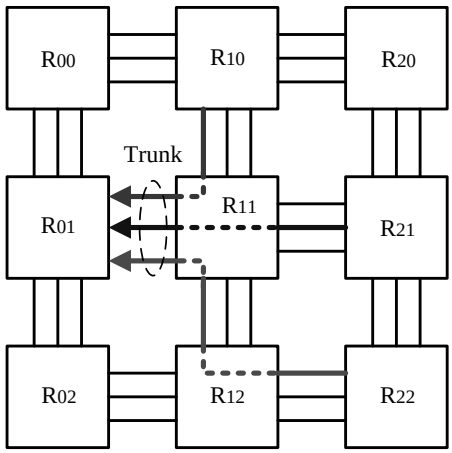
Analysis of NoC routers presented in (Dally and Peh, 2001; Mello et al., 2005; Janarthanan, 2008; Mullins, West and Moore, 2004) allows us to conclude that usage of VC in wormhole NoCs increases the saturation threshold value by no more than 20% of network capacity in comparison with wormhole NoCs without VC (taking into account the constant buffer volume per port for both cases). There is a limit, when NoC saturation threshold almost does not depend on further increase of the number of VCs. In (Mello et al., 2005) it is shown that if one increases the number of VC from 2 to 4, the saturation threshold increases from 23% to 25% of the NoC capacity. In other words, increase of number of VCs by 100% (with corresponding growth of hardware volume) leads to increase of the throughput by only 2%. Similar problems with scalability are discussed in (Mullins, West and Moore, 2004), where doubling of VC quantity per input port (from two to four) does not lead to increase of saturation threshold. In our opinion, this problem can be explained by the fact that the throughput of each physical connection is divided between VCs, which use it. This in turn leads to an increase in the period of flits transmission in corresponding logical connections.

Therefore one can make a conclusion that if we increase the number of VCs, a single physical connection between topologically neighboring routers become a bottleneck of NoC and limits the further increase of NoC saturation threshold and its throughput value.

3 Architecture of a NoC router with LAG

As shown above, the advantage and disadvantage of the VC technology is a division of a physical communication line into logical connections. Using the so-called inventive method of inversion (Altshuller, 1999), we proposed in (Korotkyi and Lysenko, 2011) an "inverse" concept – incorporation of several physical links (PL) in an aggregated logical channel (trunk). In other words, in order to link topologically adjacent routers, we offer to use several space-separated channels, each being used for transmission of individual packets. Figure 1 shows how aggregation of three PLs eliminates HOLB at the western output of router R_{11}. In the work (Korotkyi and Lysenko, 2011) it is shown that LAG helps to decrease HOLB and significantly increase NoC saturation threshold. Moreover, if we know spatial distribution of data flows in NoC, we can completely eliminate HOLB by properly adjusting the number of PLs in each trunk.

Figure 1 Eliminating of HOLB in NoC by aggregation of 3 physical links

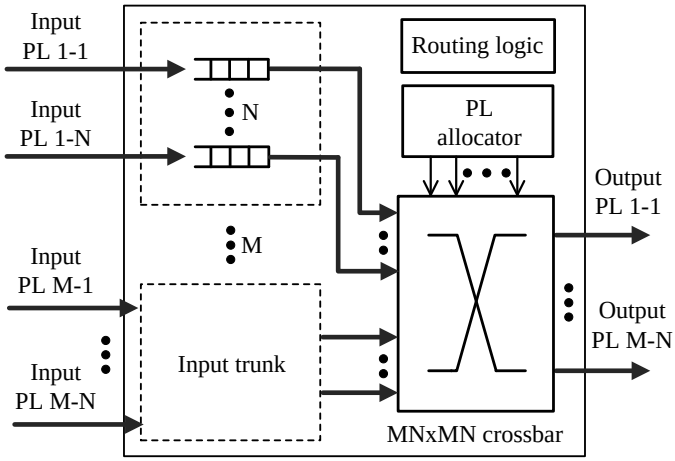


Analysis of literature shows that aggregation of physical links has its applications in macro-networks, and some documents (IEEE Standart 802.3ad, 2000; IEEE Standart 802.1ax, 2008) standardize this approach for Ethernet. Since the design solutions used in macro-networks are not suitable to NoCs because of their complexity and hardware volume, it becomes important to create a simple and efficient mechanism for implementing of LAG in NoCs. In this paper we propose one possible solution of this problem.

3.1 Structure

The block diagram of a NoC router with LAG is shown in Figure 2. Instead of the term "port", to describe the inputs and outputs of this device, we use the term "trunk", whose definition was described above. The suggested solution includes M input trunks and M output trunks, each trunk contains N PLs. The connections between PLs are established by MN×MN crossbar switch. For each input PL the incoming flits are buffered in a corresponding queue, since when the number of PLs in trunk is less than the quantity of data flows passing through it, the immediate service of arrived flits is not guaranteed.

Figure 2 Block diagram of a NoC router with LAG. PL – physical link. M – number of input-output trunks. N – number of PLs per trunk.



The data path in the suggested router takes form of a two-stage pipeline. For the head flit the pipeline latency is at least two clock cycles and for all consequent flits until the end of the packet this latency is at least one clock cycle. At the first stage of pipeline, the routing and PL allocation are performed. During the second stage, flit is extracted from the queue and traversed through the crossbar and inter router connection to the next network node. Reliable data delivery, without packet drops, is attained by "credit-based" flow control (Dally and Towles, 2004). Let us consider the above operations in more details.

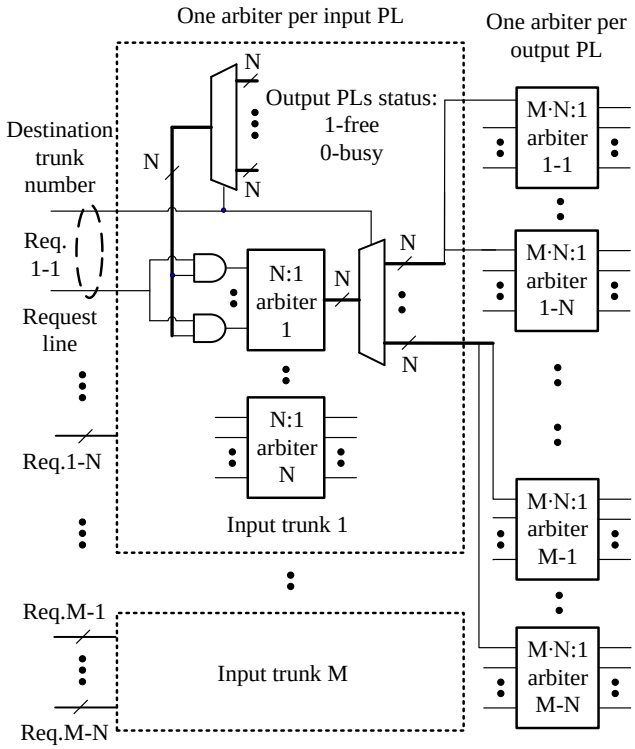
3.2 Routing

In order to reduce the device latency we use the "look-ahead" routing (Dally and Towles, 2004), when the direction of the flit transfer for the current router is determined in the previous network node. This approach allows to start establishing of connection immediately after arrival of the first flit of a packet and simultaneously determine a destination trunk for the next router. The term "destination trunk" means an output aggregated channel, which is used to accept all flits of a packet. Therefore, the routing information for the current router is contained in a special spot of the first flit of a packet, and is available at the beginning of the clock cycle.

3.3 PL allocation and switching

To move the data over the router, one needs to connect a PL at the input with a PL at the output. In the proposed device, the connection between the input and output is setup upon arrival of the first flit and is disconnected after sending of the last part of a packet. At the moment of disconnection, the corresponding PL at the output becomes free and again is ready to accept any other input PL, which wins the arbitration. After arrival of the first flit of a packet, each of the input PLs tries to get the available PL in the destination trunk by sending a request to the corresponding input of a PL allocator (Figure 3). In case of successful PL allocation, the request mentioned above is cancelled and connection between the corresponding PLs at the input and output of a router is established with the aid of crossbar switch. The number of inputs in the PL allocator is

Figure 3 Block diagram of the PL allocation module. PL – physical link. N – number of PLs per trunk. M – number of input-output trunks.



equal to the number of input PLs in a router, and each input consists of a request line and a bus, which contains a destination trunk number. As shown on Figure 3, PL allocator includes two levels of arbitration. For each input PL, the corresponding first level arbiter makes a preliminary selection of a free PL in the destination trunk. The status of output PL becomes zero (busy), when the link is allocated to any input PL. The status becomes one (free), when the last flit of a packet exits the router via busy output PL. The information about status of PLs in destination trunk comes to corresponding first stage arbiter via dedicated multiplexer (one per arbiter). The outputs of the first level arbiters are connected to the inputs of the second level arbiters via demultiplexers (Figure 3). The *k*-th output of the first level arbiter corresponds to the *p*-th input of the *q*-th second level arbiter. The values of *p* and *q* can be determined according to the following expressions:

$$p = N \cdot i + j\,, \quad q = N \cdot t + k\,,$$

Where values of *i* and *j* denote the numbers of input trunk and input PL of the corresponding first level arbiter. N characterizes the quantity of PLs per trunk. Indexes *t* and *k* define a destination trunk number and output PL number (which is previously allocated by the first level arbiter). It is possible that two or more first level arbiters choose the same output PL and output link will be contested by several inputs. Therefore, the final selection of input PL, which will be connected to the particular output PL, is made at the second level of arbitration. Here every output PL corresponds to arbiter with the number of inputs, which is equal to the number of input PLs. The active output of

second level arbiter indicates the number of input PL to which particular output PL is allocated. All the arbiters are realized based on the matrix method, providing a better throughput compared to round-robin solution (Fu and Ling, 2010).

3.4 Flit advancing

At the second stage of pipeline, all of the per-packet processing is complete and all remaining control is the flit-by-flit crossbar traversal and operations with credits. The head flit starts this process, but is handled no differently than any other flit. In order to pass a flit over the established connection, two conditions are to be satisfied: the data available in the respective queue at the input and a non-zero number of credits at the router's output PL. Then the flit will be extracted from the queue and, by the established connection, comes to the downstream router. At the same time a credit is directed to the preceding node of the network, meaning that in the input queue a new free place has appeared. Simultaneously, the number of credits for output PL, through which the flit was sent, is decreased by one.

3.5 Simulation and synthesis results for RTL model of a NoC with LAG

In order to estimate the latency and throughput of a wormhole NoC with LAG, we created the RTL model of such a NoC and made simulation of its performance. The language of model description is System Verilog. The analysis of simulation results in (Korotkyi and Lysenko, 2011) shows that usage of a LAG allows to increase NoC saturation threshold by 300% compared to classical wormhole approach (without VC). In comparison with a NoC based on VC, the proposed method leads to increase of the saturation threshold by 126% (NoC Netmaker, *Netmaker*, online), and by 152% (Hermes VC NoC, Mello et al., 2005). Mentioned paper (Korotkyi and Lysenko, 2011) contains the detailed description of experiment, which is a basis for comparison. Here we indicate only the main results. In addition, our design gives a four-time decrease in transport latency in comparison with Hermes NoC.

For clear comparison, all three investigated NoCs have 8x8 mesh topology and equal bisection bandwidth. The design parameters of the routers that used during experiments are presented in Table 1. The first two columns of Table 1 is a comparison between our design and a HERMES NoC, the last two columns refer to a comparison between a Netmaker VC and a NoC with LAG. All VC routers involved in comparison have 5 input/output ports and the LAG router contains 5 input/output trunks.

As shown in Table 1, in order to achieve the same bisection bandwidth, we chose the width of VC router's port the same as the width of the trunk in a router with LAG. Hence, the capacity of each PL inside the trunk is determined by the following formula:

$$\text{PLcapacity (in bits)} = \frac{\text{The port width in the comparable VC router (in bits)}}{\text{The number of PLs within a trunk}}$$

According to design of our router, the flit width cannot be less than 11 bit, so the trunk width in the router with 2 PLs is 22 bit. On the other side, the flit width for a HERMES VC is 16 bit and cannot be changed, because this NoC is not an open source. This fact slightly violates the requirement for equality of the port width and trunk width, but in case of Netmaker NoC, there is no problem.

Table 1 The design parameters and results of synthesis for routers used in the latency and throughput comparison between VC and LAG architectures of a NoC

Router type	HERMES, VC	Our design, LAG	Netmaker, VC	Our design, LAG
FPGA device	Virtex 2	Stratix 4	Stratix 4	Stratix 4
Number of VC per port	2	–	2	–
Number of PL per trunk	–	2	–	2
PL width, bit	–	11	–	16
Port or trunk width, bit	16	22	32	32
Buffer length, flit	8	4	4	4
LUT's	1377	1900	2400	2167
REG's	327	894	2232	1074
Fmax, MHz	–	200	130	190

From equality of the bisection bandwidth for compared architectures, we can conclude that obtained high results follow from the fact that in a NoC with LAG, each trunk passes multiple data streams simultaneously, and this reduces the probability of HOLB and increases the saturation threshold.

The hardware costs of the routers for the compared NoCs are shown in Table 1. The results for our design are obtained by its synthesis in Quartus 9.1. As a target FPGA we used Altera's Stratix IV chip. Since a Netmaker VC NoC has open source code, we adopt it to synthesize in Quartus. In a Netmaker's router, the unrestricted VC allocation and switch allocation are performed consequently in one clock cycle. More information about implementation of a Netmaker VC router can be found in (Mullins, West and Moore, 2004). In our router, the operation of PL allocation is also performed in one clock cycle and if we will pipeline it, the maximum clock frequency grows up by 60 MHz. The information about hardware costs of HERMES VC NoC was taken from (Mello et al., 2005). As shown in Table 1, the hardware costs of a NoC with LAG and its VC analogs are of the same order.

4 Justification the selection of description language for a high-performance model of a router with LAG

In order to minimize the hardware resources needed to implement a NoC with LAG in (Korotkyi and Lysenko, 2011), we proposed to make a parametric optimization of the number of PLs inside the aggregated logical connections of a NoC. Performing such optimization requires a high-performance model of investigated object. In this section and next sections, we will describe the process of creation of such a model.

It was shown in (Black and Donovan, 2004) that regardless of a language used for IC description (SPICE, Verilog, VHDL, or SystemC), the simulation time depends mainly on the extent of the detailing of description. In case of discrete-event modelling, the simulator makes the following steps: data transfer, modification of the queue of scheduled events, and performance of operations over the data. If the number of modules, intermodular connections, and data transmitted increase, the number of the required operations also increases, and the operation with the queue of scheduled events becomes more complex, obviously this leads to the growth of a simulation time. Thus, in order to

speed up the simulation, one needs to simplify the description of a model, and reduce the number of modules and intermodular links. In order to avoid the accuracy losses in simulation, one should exclude the details, which don't have significant effect on the simulation results. This becomes possible due to proper application of the method of abstracting.

To reduce the simulation time of a model by means of increasing its abstraction level, one can use any hardware description language (HDL), but SystemC has the following advantages:

- SystemC has free and open source code.
- SystemC is implemented as a library of C++ classes, this allows to use all object-oriented programming tools of this language for increase model's abstraction level.
- Possibility for application in models of a large array of existing effective and elaborated solutions implemented in C and C++, e.g. STL and Boost (*Boost C++ libraries*, online).
- Possibility for synthesis of hardware solutions, both with low-level and high-level description, e.g. Catapult (*Catapult C synthesis*, online).
- Possibility for co-simulation with code created in other HDL (Verilog, System Verilog, VHDL).
- Modern simulators, like ModelSim, compiling a model described with the aid of some HDL, into a machine-independent object code, which then is optimized and simulated. If at least one parameter of a model was changed, a new cycle of compilation and optimization is required, which takes from several minutes up to several tens of minutes. Description of IC in SystemC allows to create dynamically parameterized modules at runtime, without re-compilation and re-optimization of the model.
- During performing parametric optimization of a SystemC model, simulation and optimization are done in a single program written in C++. There is no need in transmission of a large volume of intermediate information between the optimizing and simulating parts, for example, as in case of ModelSim.

5 Implementation of a high-performance behavioral model of a NoC router with LAG

Parametric optimization of the PLs quantity in aggregated channels of a NoC, requires information about packet transfer delays and utilization rates of the trunks. There is no need in information about timings, so one can use a behavioral model, which implements an algorithm of the functioning of the device without taking into account the propagation delays of signals within IC.

5.1 General approach

During creation of high-performance SystemC model of a NoC router with LAG, we used the following rules:

- Usage of built-in C++ types instead of types related to SystemC leads to reduction of simulation time. For example, a two-symbol type *bool* is better than 4-symbol type *sc_logic* (‘0’, ‘1’, ‘x’, ‘z’). Multibit signals should be described with the type *int* and data structures, instead of arrays of one-bit variables (Black and Donovan, 2004).
- In SystemC, processes of SC_METHOD type are simulated faster than processes of SC_THREAD type, because the latter have their own stack and local variables, which demand additional operations for managing of the thread context (Black and Donovan, 2004).
- In SystemC, for transmission of information regarding signal modification, from one process to another via corresponding ports and connection, one needs to call three functions and perform three copy operations. Minimization of intermodular links on the RTL abstraction level leads to a decrease of number of calls of these functions and, therefore, it leads to a shorter simulation time (Black and Donovan, 2004).
- In order to reduce the number of modules and intermodular links on the RTL abstraction level, each router in a NoC with LAG is described with process of type SC_METHOD. This process has interfaceses for signal connection. All submodules inside a router, and all links (signals) between them, are replaced by a sequential program, which realizes an algorithm of a router operation. This program uses built-in C++ types and high-level containers. For example, for implementation of fifo-buffers of a router, one can use type *deque* from STL.

5.2 Data types and variables

Let us consider an algorithm of the program, which simulates a process of operation of a NoC router with LAG. Begin with reviewing the types of used variables. Firstly, we have input and output interfaces of a router – *clk, rst, cntrl_in, cntrl_out, flits_in* and *flits_out*. The inputs *clk* and *rst* have a type of *sc_in<bool>*. The procedure, which simulates a router operation, is called on the rising edge of *clk* signal. The input *rst* serves to reset and initialization of a model. The input *cntrl_in* represents an array of variables of *sc_in<bool>* type, with dimension $M \times N$, where N is the number of PLs in each of M trunks connected to a router. Each element in *cntrl_in* array represents a flow control input, which accepts the credits from a downstream router (data receiver). Everything said above is also true for *cntrl_out*, taking into account that the elements of this array are flow-control outputs and have a type of *sc_out<bool>*. The input interface, which accepts the incoming flits, is array, entitled *flits_in*. This array has dimension $M \times N$ and its entries have a type of *sc_in<flit_t>*. The structure *flit_t* has the next fields: *valid (bool), head (bool), tail (bool), output_trunk (int)*, field of transmitted data, and also the fields for additional routing information, which are not essential for explanation of this algorithm. Dimension of array *flits_out* is $M \times N$ and its entries have a type of *sc_out<flit_t>*. The other variables used by the program, are $M \times N$ arrays and will be considered during the algorithm description. Basically elements of all mentioned arrays have a type of *bool*, except *allocated_pl (int), credit_count (int),* and *output_trunk (int).*

5.3 The algorithm

Let us consider the steps of the operational algorithm of high performance model of a NoC router with LAG. The procedure, which implements the algorithm, is called on the positive edge of signal *clk*. Algorithm includes the following steps:

1. If the signal *rst* is equals to *true*, one performs the procedure, which makes a primary initialization of variables, otherwise one goes to the next step. The initialization procedure resets to zero (*false*) all variables, which are used in the program, except of arrays *pl_status* and *credit_count*. All elements of array *pl_status* should be setup to *true*, and all elements of array *credit_count* should be equal to the dimension of queues in the system.
2. $\forall i, j : i \in [1; M], j \in [1; N]$, where j defines PL in the i-th input trunk, if *input_trunk_pop[i][j]==true* and bit *tail* of the flit at the head of queue *fifo[i][j]* equals *true* (the last flit of the packet), perform *allocated_pl_valid[i][j] = false.*
3. $\forall i, j : i \in [1; M], j \in [1; N]$, where j defines PL in the i-th input trunk, if *input_trunk_pop[i][j]==true*, perform removing of the flit from the head of queue *fifo[i][j]*.
4. $\forall k, p : k \in [1; M], p \in [1; N]$, where p denotes PL in the k-th output trunk, if *flits_out_tail[k][p]==true* and *flits_out_valid[k][p]==true*, then mark the respective output channel as non-occupied (free) by performing *pl_status[k][p] = true*.
5. $\forall i, j : i \in [1; M], j \in [1; N]$, where j corresponds to physical link in the i-th input trunk, if the queue *fifo[i][j]* is not empty, *credit_count[k][p]!=0* and *allocated_pl_valid[i][j]==true*, then assign *true* to *input_trunk_pop[i][j]*, else – *false*, where *k=out_trunk[i][j]*, *p=allocated_pl[i][j]*. At this step we determine what queues are ready for extracting flits and transferring them to the downstream routers during the next clock cycle.
6. $\forall i, j : i \in [1; M], j \in [1; N]$, where i and j represent indexes of the signal of outgoing credit for the j-th PL in the i-th input trunk, perform *cntrl_out[i][j]= input_trunk_pop[i][j]*.
7. $\forall i, j : i \in [1; M], j \in [1; N]$, where j is indexing the physical connection in the i-th input trunk, if *allocated_pl_valid[i][j]==true*, then perform the following substeps:
   7.1. Copy the flit, located at the head of queue *fifo[i][j]*, in variable *tmp_flit*.
   7.2. If it is the first flit of the packet (*tmp_flit.head==true*), execute a look-ahead routing, and thus define the destination trunk for the next node of the network by means of calling function *route(…)*: *tmp_flit = route(tmp_flit)*. This operation renews the routing information for variable *tmp_flit*. Realization of a look-ahead routing is described in (Dally and Towles, 2004).
   7.3. Modify the *valid* bit of variable *tmp_flit*: *tmp_flit.valid = input_trunk_pop[i][j];*
   7.4. Write the variable *tmp_flit* into output interface *flits_out[k][p]*: *flits_out[k][p] = tmp_flit.* Here *k* and *p* are calculated by formulas for step 5.
   7.5. Execute *flits_out_valid[k][p] = tmp_flit.valid*. Here *k* and *p* are calculated by formulas for step 5.
   7.6. Execute *flits_out_tail[k][p] = tmp_flit.tail*. Here *k* and *p* are calculated by formulas for step 5.

7.7. If *tmp_flit.valid==true*, execute: *credit_count[k][p] = credit_count[k][p] – 1*.

Here *k* and *p* are calculated by formulas for step 5.

8. $\forall i, j : i \in [1; M], j \in [1; N]$, where *j* is indexing the PL in the *i*-th input trunk, if *flits_in[i][j].valid==true*, then copy the flit from input interface *flits_in[i][j]* into tail of the queue *fifo[i][j]*.

9. $\forall i, j : i \in [1; M], j \in [1; N]$, where *j* is indexing the PL in the *i*-th input trunk, if in the head of the queue *fifo[i][j]* we have the first flit of the packet (whose *head* bit takes the *true* value), retain the routing information:

out_trunk[i][j] = fifo[i][j].front().output_trunk;

10. $\forall i, j : i \in [1; M], j \in [1; N]$, where *j* is indexing the physical connection in the *i*-th input trunk, form requests for allocation of PLs in output trunks. To do this, perform the following: if the queue *fifo[i][j]* is not empty and *allocated_pl_valid[i][j]==false*, then assign the *true* value to the element of array *alloc_req[i][j]*, otherwise set it equal to *false*.

11. Call function *allocate(…)*, which realizes the algorithm of functioning of the module, which is responsible for allocation of PL (Figure 3). The purpose of *allocate(…)* function consists in providing of a free PL in output trunk to the each input PL that have made a request at the previous step of the algorithm. The input data of this function are arrays: *alloc_req*, *out_trunk* and *pl_status*; output data are arrays: *allocated_pl*, *allocated_pl_valid* and *allocated_pl_out*. For each input PL, the element of array *alloc_req* determines a request for obtaining of a PL in the output trunk. An element of array *out_trunk* defines the number of output trunk assigned to the given input PL. An element of array *allocated_pl* defines the number of a PL allocated; if the value of an element in array *allocated_pl_valid* is equal to *true*, the allocation of a PL for a given input was successful (as a result of executing the function *allocate(…)*at the current step), otherwise *allocated_pl_valid==false*. For each output PL, the element of array *pl_status* shows the status of a PL (*true* – available for allocation, *false* – busy, i.e. is allocated for any input PL during previous steps). If the value of an element in array *allocated_pl_out* is *true*, this means that this output PL was allocated for any of input PLs due to execution of function *allocate(…)* during the current step.

12. $\forall k, p : k \in [1; M], p \in [1; N]$, where *p* denotes a PL in the *k*-th output trunk, if *allocated_pl_out[k][p]==true*, indicate the respective output channel as busy by implementing *pl_status[k][p] = false*.

13. $\forall k, p : k \in [1; M], p \in [1; N]$, where *k* and *p* perform indexation of input signal of the credit corresponding to PL *p* in *k*-th output trunk, if *cntrl_in[k][p]==true*, then execute *credit_count[k][p] = credit_count[k][p]*+1.

6 Simulation results

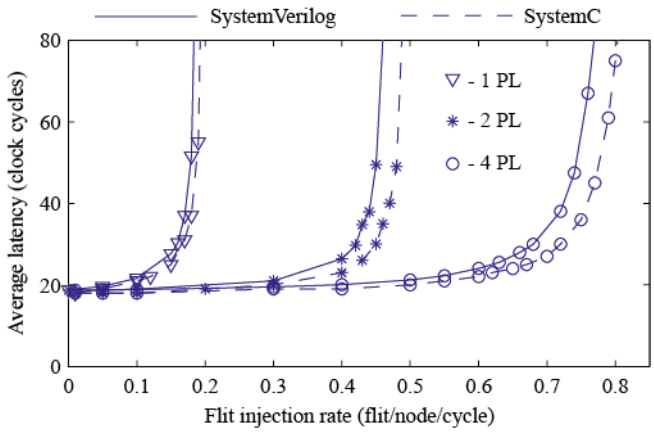
In this section we estimate the latency-throughput and performance characteristics of the proposed high-level behavioral model for a NoC with LAG. Based on SystemC model of a router with LAG, described in previous section, we created a NoC with 8x8 mesh topology. Every node in this network is connected to the neighbor routers via four bilateral trunks in accordance with the selected topology. Particularly, each router has 5 input/output trunks. The fifth pair of trunks is used for connection of a router to a traffic generator-receiver, which imitates a computational module. The length of the router's input queues is four flits per PL. The compilation of SystemC library and the described NoC's model were performed using a g++ compiler, which is a part of GCC version 4.3.4. The execution of the compiled model was conducted in Cygwin 1.7.7 environment on a PC with Windows XP SP3, Core 2 Duo T5300 processor and 2 GB of RAM inside.

For the standardization purposes, we chose a simple dimension ordered XY routing, which is widely used in NoCs. During a simulation run, the traffic generators create packets, which consist of five flits each, and insert these flits into the network with intensity λ (flit injection rate). The value λ is in the range from 0 to 1, and shows the average number of flits injected in a network by a computational node during the clock cycle. The time intervals between flit injections are distributed according to the exponential law and are equal, in average, to $1/\lambda$. In order to keep the time distribution characteristics, the created packets, before injection, are buffered into source queues (Dally and Towles, 2004). The traffic generators inject the flits in the NoC via only one PL. The traffic receivers read all incoming flits via all PLs of a corresponded trunk. In order to describe the spatial distribution of packets, a uniformly random law was selected, when each source can send the data to each destination at the same probability (except sending to itself). For each following packet, a new destination point is generated. After start of simulation, all nodes in a network generate 1100 packets, and the results pick-up at each traffic sink starts after receiving of the first 100 messages. This procedure is for transmission of the NoC in steady-state condition (Dally and Towles, 2004). The simulation is over after registration of all sent messages in the destination points.

Figure 4 shows the dependence of a NoC's transport delay (in clock cycles) on a flit injection rate for different configurations of the high-level SystemC model of a NoC with LAG and its synthesizable System Verilog counterpart. The datasets for mentioned SystemVerilog model were obtained by us and were described in (Korotkyi and Lysenko, 2011). For each model, three configurations of a NoC were investigated: with one, two and four PLs per trunk.

As shown on Figure 4, before the sharp growth of the curves of traffic delay as a function of flit injection rate, the behaviors of both models are the same (the error is less 1%). At the moment of the network saturation, the error of the high speed SystemC model grows dramatically, which is based on the exponential type of dependence shown in Figure 4. The range of the flit injection rate, where the error of estimation of the transport latency increases by 50% or more, determines an accuracy of prediction of the network saturation threshold, and is 5.6% of the saturation threshold for the NoC configuration with one PL per trunk, 6.1% for the NoC with two PLs per trunk and 3.9% for the NoC with four PLs per trunk. We think that the decrease of a simulation accuracy of the high level SystemC model is due to the increase of its abstraction level. However, this model can be used during the initial steps of parametric optimization of a NoC, and

Figure 4 The dependency of a NoC latency on a flit injection rate for the synthesizable RTL model based on SystemVerilog, and its high level counterpart based on SystemC.



with followed verification of the obtained results based on a more precise model of the gate level of abstraction.

The average simulation time and the amount of memory used for various configurations of the investigated models are shown in Table 2. For a given number of packets to be injected into a network during a single model run, the simulation time depends on a flit injection rate, and decrease with the growth of the latter. The values of the simulation time in Table 2 were obtained by averaging of the results of model runs for different values of the flit injection rate (from zero up to the saturation threshold) for the same network configuration. The measurements of the execution time of a single model run were done using a utility *time*, which is a part of a Cygwin package. The measurements of the amount of memory occupied by the model were done using a Windows Task Manager.

Table 2 shows that the proposed high-level SystemC behavioral model gives a significant reduction in simulation time (from 3.9 to 10.2 times), and the amount of occupied memory (from 29.4 to 121 times) in comparison with its RTL counterpart based on System Verilog.

Table 2 Simulation time and memory usage

	Model characteristics					
	1 PL/trunk		*2 PL/trunk*		*4 PL/trunk*	
	Time, sec.	*Mem., Kbytes*	*Time, sec.*	*Mem., Kbytes*	*Time, sec.*	*Mem., Kbytes*
System Verilog	1 061	139 928	1 102	278 740	1 710	714 728
SystemC	268	4 760	146	5 156	167	5 908
gain	3.9	29.4	7.5	54	10.2	121

7 Conclusions and outline of future research

This paper describes investigation of our high-performance SystemC model of a router for a NoC with LAG. The results of simulation show that for our model, the prediction error of the NoC saturation threshold does not exceed 6.1%, in comparison with its RTL counterpart based on SystemVerilog. Herewith, the simulation time is reduced in 10 times, and the amount of used memory decreased in 121 times. These results were achieved by reduction the number of modules on a hardware level of abstraction and connections between them, using of built-in C++ data types, and also using of SystemC processes like SC_METHOD instead of SC_THREAD.

Our next step is to develop a method of parametric optimization of the number of PLs in aggregated logical connections of a NoC with LAG based on the developed high-performance SystemC model. The priority direction of research is a search for reduction of an error for the developed fast SystemC model in comparison with its RTL SystemVerilog counterpart.

References

Angiolini, F., Meloni, P. and Benini, L. (2007) 'A layout-aware analysis of networks-on-chip and traditional interconnects for mpsocs', IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, Vol. 26, No. 3, pp. 421-434.

Atienza, D., Angiolini, F., Murali, S., Pullini, A., Benini, L. and Micheli G.D. (2008) 'Network-on-chip design and synthesis outlook', Integration The VLSI journal, Vol. 41, No.3, pp. 340-359.

Altshuller, G. (1999) The innovation algorithm. TRIZ, systematic innovation and technical creativity, Technical innovation centre, Worcester, MA.

Boost C++ libraries. [online] Available at http://www.boost.org/ (Accessed 12 February 2012).

Black, D. and Donovan, J. (2004) SystemC: from the ground up, Kluwer Academic Publishers, Boston, USA.

Bjerregaard, T. and Mahadevan, S. (2006) 'A survey of research and practices of network-on-chip', ACM Comp. Surveys, Vol. 38, No.1, pp. 1-51.

Catapult C synthesis. [online] Available at http://www.mentor.com/esl/catapult/ (Accessed 12 February 2012).

Dally, W.J. (1990) 'Performance analysis of k-ary n-cube interconnection networks', IEEE Transactions on Computers, Vol.39, No.6, pp. 775-785.

Dally, W.J. (1992) 'Virtual-channel flow control', IEEE Transactions on Parallel Distributed Systems, Vol.3, No.2, pp. 194-205.

Dally, W.J. and Towles, B.P. (2001) 'Route packets, not wires: on-chip interconnection networks' Proceedings of the 38th annual Design Automation Conference, Las Vegas, USA, pp.684-689.

Dally, W.J. and Towles, B.P. (2004) Principles and Practices of Interconnection Networks, Morgan Kaufmann Publishers, San Francisco.

Dally, W.J. and Peh, L.S. (2001) 'A delay model and speculative architecture for pipelined routers' in Proceedings of International Symposium on High-Performance Computer Architecture, Nuevo Leone, Mexico, pp. 255–266.

Fu, Z. and Ling, X. (2010) 'The design and implementation of arbiters for networks-on-chip', in Proceedings of 2nd International Conference on Industrial and Automation Systems, Dalian, China, pp. 292-295.

Gu, H. (2011) 'Survey of dynamically reconfigurable Network-on-chip', in Proceedings of International Conference on Future Computer Sciences and Application, Hong Kong, China, pp. 200-203.

Janarthanan, A. (2008) Networks-on-chip based high performance communication architectures for FPGAs. Ph.D. thesis, Department of Electrical and Computer Engineering and Computer Science, University of Cincinnati, Cincinnati, Ohio, USA.

IEEE Standart 802.1ax (2008) IEEE Standart for Local and Metropolitan Area Networks – Link Aggregation.

IEEE Standart 802.3ad (2000) Link aggregation.

Korotkyi, I. and Lysenko, O. (2011) 'Hardware implementation of link aggregation in networks-on-chip', in Proceedings of World Congress on Information and Communication Technologies, Mumbai, India, pp.1112-1117.

Marculescu, R., Chang, N., Ogras, U.Y. and Lee, H.G. (2007) 'On-chip communication architecture exploration: a quantitative evaluation of point-to-point, bus and network-on-chip approaches', ACM Transactions on Design Automation of Electronic Systems, Vol.12, No.3, pp.1-20.

Marculescu, R. and Bogdan, P. (2009) 'The chip is the network: toward a science of network-on-chip design', Foundations and Trends in Electronic Design Automation, Vol.2, No.4, pp.371-461.

Marculescu, R., Ogras, U., Peh, L., Jerger, N. and Hoskote, Y. (2009) 'Outstanding research problems in NoC design: system, microarchitecture, and circuit perspectives', IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, Vol.28, No.1, pp. 3-21.

Mello, A., Tedesco, L., Calazans, N. and Moraes, F. (2005) 'Virtual channels in networks on chip: implementation and evaluation on Hermes NoC' in Proceedings of 18th Symposium Integrated Circuits and System Design, New York, USA, pp. 178-183.

Mullins, R., West, A., Moore, S. (2004) 'Low-latency virtual-channel routers for on-chip networks', in Proceedings of 31-th International Symposium on Computer Architecture, Munich, Germany, pp. 188-197.

Netmaker, [online] Available at http://www-dyn.cl.cam.ac.uk/~rdm34/wiki/ (Accessed 12 February 2012).