# System Level Modeling of Dynamic Reconfigurable System-on-Chip

Elena Suvorova, Nadezhda Matveeva, Alexey Rabin, Valentin Rozanov
Saint-Petersburg State University of Aerospace Instrumentation
Saint-Petersburg, Russian Federation
suvorova@aanet.ru, {nadezhda.matveeva, alexey.rabin, valentin.rozanov}@guap.ru

***Abstract*—In this paper methods of dynamically reconfigurable multi-core System-on-chip (SoC) design are discussed, the approaches of system modeling for evaluation of these systems are presented. The dynamically reconfigurable SoC can be developed using the FPGA and the ASIC technologies. The implementations of dynamic reconfiguration using these approaches are essentially different. The system level modeling is used to evaluate the performance of dynamically reconfigured systems in the early stage of their development. The models of dynamically reconfigurable systems have very significant differences from the models of systems without a dynamical reconfiguration. The development of such models may require extensions of existing tools and specification of mechanisms functionality. In this paper the existing tools for SoC system design and the requirements for it to allow modeling of reconfigurable systems are considered. We propose mechanisms for system level modeling of the dynamically reconfigurable Networks-on-Chip (NoC) implemented on the ASIC technology.**

## I. INTRODUCTION

The reconfiguration of the system can be implemented on a static or a dynamic level [1].

The static reconfiguration (often referred to the compile time reconfiguration) is the simplest and most common approach for implementing design with a reconfigurable logic. Hardware resources remain static for the life of the design. Static reconfiguration is possible only on the Register Transfer Level (RTL) development stage. The dynamic reconfiguration (often referred to the runtime reconfiguration) uses a dynamic allocation scheme that reallocates hardware at a runtime. It can increase the system performance using highly optimized circuits that are loaded and unloaded or reconfigured dynamically during the operation of the system [1].

We consider the system level modeling of the dynamic reconfigurable SoC in this paper.

The main problems with the system modeling of the dynamic reconfigurable systems are:

1) the validation of reconfigurable systems behavior during the functioning;
2) the validation of reconfigurable systems behavior during the reconfiguration;
3) the evaluation of the task execution time in a reconfigurable system;
4) the preliminary evaluation of energy consumption during functionality and reconfiguration;
5) the evaluation of a resource utilization for a reconfigurable system.

The tasks 1, 3, 4, 5 are typical not only for the dynamically reconfigurable systems. The task 2 is specific for these systems. This task is very important because most of errors arise in such systems by the reason of transient processes during a reconfiguration.

Therefore, it is important that the reconfiguration process can be modeled with a high degree of detail for these systems. The degree of detail for the simulation of reconfiguration process should be essentially more than for the simulation of the static functionality. We consider approaches to simulation of system reconfiguration process in this paper.

The dynamically reconfigurable systems could be implemented on the ASIC or the FPGA technologies. The major opportunities for dynamic reconfiguration provided by changing of the interconnection structure in systems implemented on the ASIC technology. However, there are some possibilities to change the logic (functionality) of components.

The dynamically reconfigurable interconnection structure can be used for implementation of NoCs with a different interconnection graph in the same platform. The interconnection structure adapts to current data flow graph [2 - 5]. Nowadays the different variants of such systems are implemented [3 - 4].

The dynamic reconfiguration of interconnections is advisable on the subblock layer [5]. This approach can be used for implementation of dynamically reconfigurable data paths. Also, it could be used for a dynamic change of the ratio of the subcomponents' number/capacity.

ISSN 2305-7254

It is also possible to dynamically change the behavior of components in the ASIC technology. Nowadays it is achieved through the use of the look-up tables or libraries with reconfigurable logical components developed for the ASIC technology. These components can be configured for implementation of different functions such as NAND, NOR, INV [6, 7].

The dynamic reconfiguration of FPGA from different manufacturers, for example, Xilinx, Virtex, Altera is performed on similar schemes. The region, wherein the CLB is disposed, is divided into frames – typically vertical columns wide several CLB (depending on the type of FPGA) [8, 9, 10]. Each of these columns can be independently reconfigured with using of a separate bitfile.

The project is divided into zones with the dynamical reconfiguration possibility and without this possibility [11].

Thus, dynamically reconfigurable systems implemented on the FPGA have a very high degree of flexibility. One component of the system can be replaced with another one having a completely different functionality. For example, RISC core can be replaced by a video codec.

Almost all of the existing system level modeling approaches for reconfigurable systems are focused on the FPGA based systems. Different programming languages, such as SystemC, SpecC and SystemVerilog are used for the specification of systems within these approaches [14].

For the simulation of communication between blocks is used the TLM-2.0 standard introduced in SystemC.

This paper is organized as follows. In paragraph 2 we consider the existing approaches to the system level simulation of the implemented on the FPGA reconfigurable systems using SystemC and TLM, their capabilities and limitations.

In paragraph 3 we suggest approaches to the system level simulation of implemented on the ASIC reconfigurable systems using the SystemC library.

## II. SystemC and TLM for simulation of reconfigurable SoC

SystemC is the ANSI standard C++ class library for system and hardware design for use by designers and architects who need to address complex systems that are a hybrid between hardware and software [12]. SystemC is widely used for a SoC design.

OSCI Transaction-Level Modeling Standard (TLM-2.0) was presented in the IEEE Std 1666™-2011. At the simplest level a TLM is a set of the SystemC modules (i.e. C++ classes), each providing one or more sockets, through which the SystemC modules may read and write data [13].

Dynamic objects creation or elimination of the SystemC module are not supported according to SystemC. They are limitations for creation Dynamically Reconfigurable Systems using SystemC. However, the SystemC standard supports dynamic processes. The function *sc_spawn* is used to create a static or dynamic spawned process instance. Spawned processes may be created by calling the function *sc_spawn* during elaboration or simulation, [12].

The function *sc_spawn* may be called during elaboration, in which case the spawned process is a child of the module instance, which function *sc_spawn* is called, or it is a top-level object, if the function *sc_spawn* is called from the function *sc_main* [12].

The function *sc_spawn* may be called during simulation, in which case the spawned process is a child of the process that has called the *sc_spawn* function. The function s*c_spawn* may be called from a method process, a thread process, or a clocked thread process [12].

If the function sc_spawn is called during the evaluation phase, the spawned process shall be made runnable in the current evaluation phase. If the function *sc_spawn* is called during the update phase, the spawned process shall be made runnable in the next evaluation phase.

The function *sc_spawn* is useful for making models of a reconfigurable system. However, this is not enough. Main items of Dynamically Reconfigurable Systems are generation or elimination while the system is running. In order to express these behaviors naturally at the system level design, modules have to be generated or eliminated dynamically and ports and channels have to be connected and dispatched dynamically [14].

There are several approaches for solving this problem. For example, authors [15] introduced an extension to SystemC in 2006. It's name is the ReChannel library. The main goal of the ReChannel library is to enable modeling and simulation of runtime reconfigurable systems – which obviously might change their hierarchy and/or interconnection characteristics during runtime – with SystemC.

This library does not depend on the SystemC kernel and has been designed to work with any SystemC simulator that conforms to the IEEE standard 1666 Open SystemC Language Reference Manual [2005]. It is very important, because a non-standard simulation kernel is a bottleneck for a design process. Compatibility with SystemC can be loosed, when a new version standard will be released. The only hard restriction is the necessity to use a compiler supporting partial template specialization.

Portal is used for this library. A portal is a special switch, designed to connect a static channel to a port of a reconfigurable module. The use of portals allows the usage of any, even a custom-built SystemC channel, in a reconfigurable context, which leads to a highly flexible methodology for the reconfigurable systems modeling [15]. The ReChannel library is free. Source files are available from the Technical Computer Science University of Bonn

site [16]. The library is distributed without any warranty. A designer can edit source files, but designers should learn sources of the ReChannel library first.

Authors [14] propose other extended SystemC library for the system level modeling and simulation of the system, which includes Dynamically Reconfigurable Architectures. The proposed library consists of a dynamic module, a dynamic port, and a channel pool. Using this extended library allows to generate and eliminate modules and ports during a simulation. It is not supported by the SystemC 2011 standard [12].

Authors [17] present a new methodology for simulation of a dynamic reconfigurable system. This methodology is based on the using of the blocked modules list. Instead of immediately executing the processes sensitive to an event, they propose the blocked modules list to be checked before its execution. The blocked modules list can change during a simulation. SystemC was selected as case study for a methodology implementation. Authors modify the SystemC kernel source code [17]. Moreover, each module has own area value. Log files are generated during a simulation. A chip area usage is generated from an execution log file. It is useful for understanding of a total chip utilization.

Methodology for Designing Partially Reconfigurable Systems is described in the paper [18]. Authors use TLM-2.0 for communications between modules and from the reconfiguration manager to the modules. The typical module is based on the SystemC dynamic threads, used to switch between tasks and change the functionality of a module during a runtime. The OSCI TLM-2.0 standard [12] supports loosely-timed and approximately-timed coding styles. The main goal of presented methodology is the modeling of the parallel behavior of the tasks inside the FPGA. Therefore, this methodology uses approximately-timed coding style, because a non-blocking transport interface is supported by it.

TABLE I. COMPARISON OF THE REVIEWED APPROACHES

| Name | Re-Channel | Dynamic Module Library | Methodology for Modeling and Simulation of Dynamic and Partially Reconfigu-rable Systems | Methodology for Designing Partially Reconfigu-rable Systems Using Transaction-Level Modeling |
|---|---|---|---|---|
| Change SystemC kernel | no | no informa-tion | yes | no information |
| Use TLM | no | no | no | yes |
| Open access | yes, free library | no | no | no |

There are two main ways to the reconfiguration implementation in existing approaches with SystemC:

- development of extended SystemC library and classes
- SystemC kernel modification

There are several versions of the SystemC standard. Current version was published in 2011. Previous version was published in 2005. Papers [14,15,17] are based on standard 2005. Current version has many important changes, which can be used for solving tasks of reconfigurable system modeling.

These changes are such as: [12]:

- new process control member functions, by which processes can suspend, resume, reset, or kill other processes or themselves;
- simulation can be paused, and there is a function to get the current state of simulation (elaboration, running, paused, stopped, and so forth);
- event lists, as passed to the functions *wait* and *next_trigger*, can be constructed as explicit objects, making it possible to create event lists containing a parameterized or variable number of events;
- a new utility class *sc_vector* makes it easy to construct vectors of modules, ports, exports, and channels and to bind vectors of ports;
- the *bind* function of the standard port and socket classes has been made virtual;
- *sc_mutex* and *sc_semaphore* are now derived from *sc_object* rather than from *sc_prim_channel*, so they may be instantiated dynamically.

The following operations are permitted while simulation is paused:

- calls to function *sc_spawn* to create dynamic processes;
- the instantiation of objects of a type derived from *sc_object* provided that instantiation of those objects is permitted during simulation;
- calls to function *sc_stop*.

The member functions of class *sc_process_handle* are described in this clause and its subclauses are concerned with process control. Examples of process control include suspending, resuming, killing, or resetting a process instance. Several of the process control member functions are organized as complementary pairs: *suspend* and *resume*; *disable* and *enable*; *sync_reset_on* and *sync_reset_off*; *kill*; *reset*, *throw_it*. With *suspend/resume*, the kernel keeps a record of whether the target process would have awoken while in fact being suspended, whereas with *disable/enable*, the kernel entirely ignores the sensitivity of the target

process while disabled. *Kill* interrupts and irrevocably terminates the target process. *Reset* interrupts the target process and, in the case of a thread process, calls the associated function again from the top.

The function *sc_get_status* shall return one of the eight values of type *sc_status* to indicate the current phase of elaboration or simulation.

The latest SystemC version does not support dynamic generation and elimination objects of class *sc_module, sc_port, sc_export*, or *sc_prim_channel* during a simulation. These features are required for modeling of dynamically reconfigurable systems, which are built on FPGA.

SystemC standard has functions of class *sc_process_handle*. They are concerned with process control. These functions are such as *suspend* and *resume*; *disable* and *enable*; *sync_reset_on* and *sync_reset_off*; *kill*; *reset*, *throw_it*. By the way *sc_mutex* and *sc_semaphore* are derived from *sc_object* rather than from *sc_prim_channel*, so they may be instantiated dynamically. Also the bind function of the standard port and socket classes has been made virtual. These features aren't enough for modeling of complex dynamic reconfigurable system. If we select an approach to create all modules and possible connections between modules before simulation, then system level model may be huge. Its modeling time will be very high. We describe some approaches which expand opportunities for system level modeling with SystemC.

## III. Approaches for development of reconfigurable NoC system level models based on ASIC

Let's consider whether there is enough opportunities for a simulation of the ASIC based dynamically reconfigurable systems using SystemC and TLM-2.0.

The dynamic reconfiguration possibilities for ASIC are significantly limited in comparison with FPGA. Typically, a set of components and possible interconnections within the ASIC is known beforehand (behavior of some components can be configurable). The reconfiguration predominantly can be implemented due to activation and deactivation of interconnections and blocks.

### *A. Simulation of dynamically reconfigured interconnection stricture between components*

As it is shown above, the dynamic reconfiguration of a communication system is achieved by inserting the selectors implemented in the form of multiplexers, or otherwise. These selectors make it possible connection of different lines to the same port at different times. The mechanism *multi_passthrough_initiator_socket* and *multi_passthrough_target_socket* can be used to support this feature. This mechanism provides binding of one source and multiple receivers, binding of multiple sources and one receiver.

Let's consider, for example, implementation of typical fragment of interconnection structure, shown in Fig. 1.

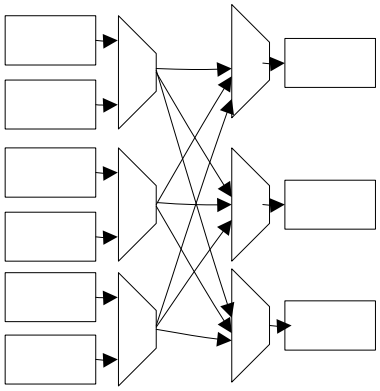

Fig. 1. The typical fragment of reconfigurable communication system for NoC implemented on the ASIC technology

The structure of TLM based model is represented in Fig. 2. In this model, instead of multiplexers a direct connection between sockets is used. A particular set of connections used at the current time depends on the system configuration.

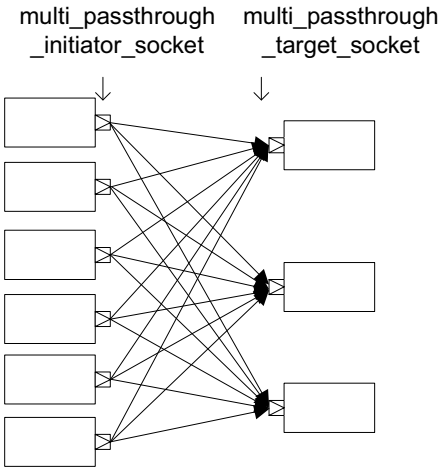


Fig. 2. The TLM model of reconfigurable communication system fragment using *multi_passthrought_initiator_socket* and *multi_passthrought_target_socket*

### *B. Simulation of dynamically reconfigured functional blocks*

The ratio of the number/capacity of subcomponents could be dynamically changed to provide dynamical reconfiguration of component's behavior.

An example of such a component is represented on Fig. 3. In this example, the RTL model of data handler includes four ALUs. These blocks are connected into chain by the carry signal. These carry signals can be used or not used in each ALU depending on the configuration.

Accordingly, the input data is handled as a single word or a group of several short words. It is advisable to use a similar structure for the system level model for detailed simulation of a reconfiguration process.

The component with minimal word size is used as basic component for system level model (as in RTL model). All components are generated statically. All possible connections between components (for example, carry lines between the ALU) implemented statically. The set of currently used connections is determined dynamically during simulation.

The data received by the handler is represented as transactions with fixed size, without interpreting of actual size and number of words in transaction. The class *simple_target_socket* is used to transmit data transactions and configuration transactions.

Two processes exist inside the handler. First of them divides received via *simple_target_socket* data to slices for every ALU. Second process assembles data slices from ALU to the output transaction. This transaction goes out via *simple_initiator_socket*.

Thus, standard SystemC features are enough for an implementation of a system level model for this reconfiguration type.

Let's consider system level simulation of reconfiguration components based on a look-up table. At the system level, the look-up table may be implemented in the form of an array. The class *sc_signal* can be used for implementation input and output signals.

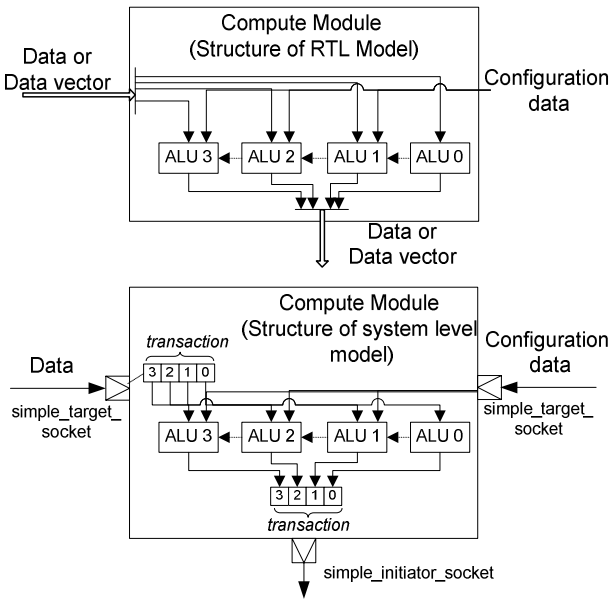


Fig. 3. The RTL model and the system level model of reconfigurable data handler

A mechanism for downloading of a new content to look-up table should be implemented. The reconfiguration command from the configuration manager can be translated via *simple_target_socket*. The content can be loaded from a file. The time for content reloading in real system takes a few cycles. This should be taken into account in the implementation of loading mechanism.

Let's consider the example – model of finite state automata for the handling of packet headers based on a look-up table. The possible variant of automata is represented on Fig. 4. This automata has 4 inputs, 4 outputs, and until 8 states. The structures of the RTL model and the system level model of this automata are shown on Fig. 5. In this way, standard SystemC features are enough for implementation of system level model for this reconfiguration type.

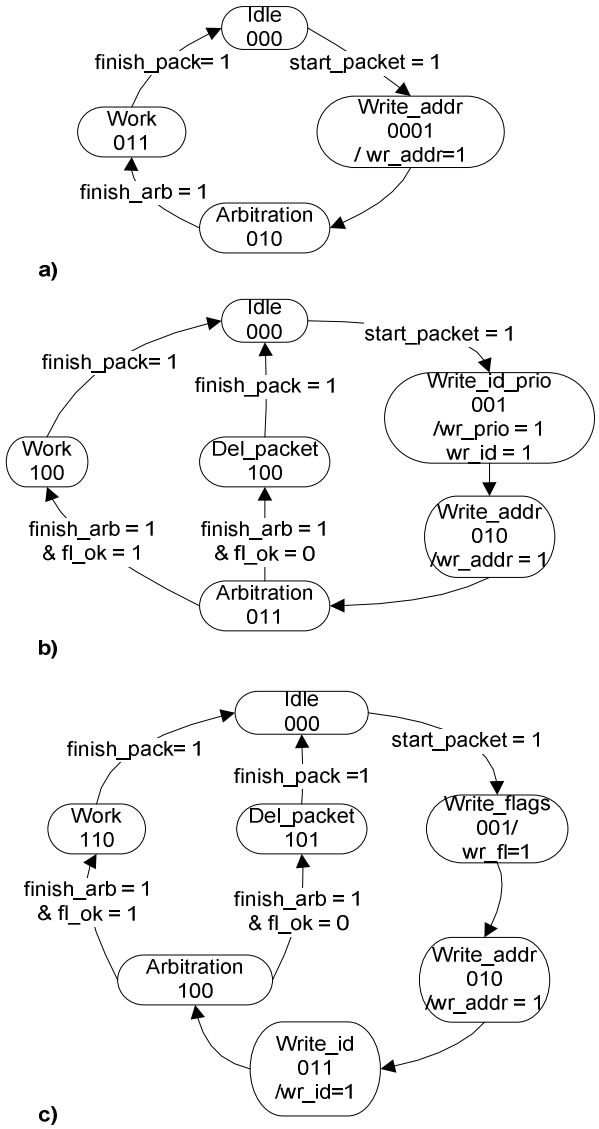


Fig. 4. The possible variants of reconfigurable automata

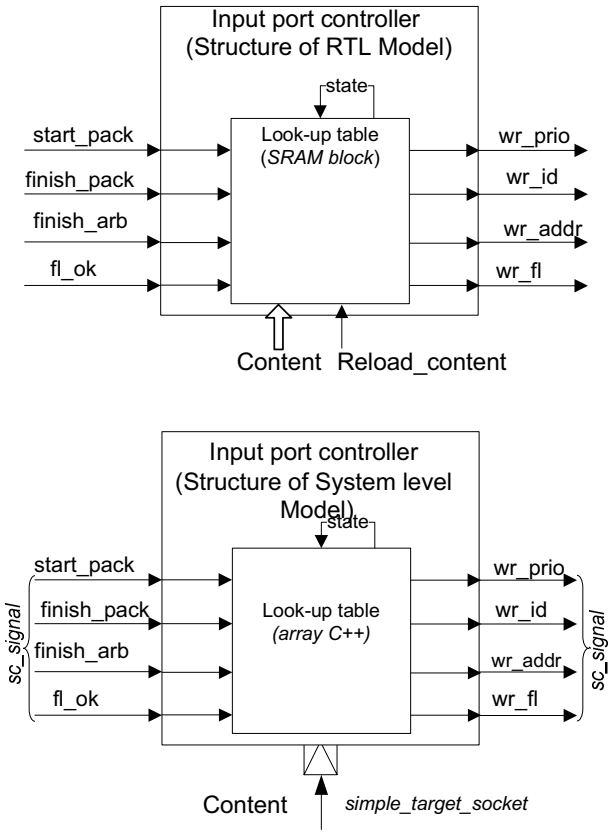


Fig. 5. The RTL and the system level models of a dynamically reconfigurable automata

The resources inside the component can be reallocated dynamically. For example, the buffer space belongs to the router port can be distributed differently between the virtual channels. The Fig. 6 illustrates an example implementation of the buffering block with two virtual channels. The amount of buffer space for every virtual channel is set dynamically.

In the RTL model a buffer is implemented on single memory block. The memory block is divided logically into subblocks. Each subblock can be assigned to the virtual channel 0 or to the virtual channel 1 depending on the amount of buffer space allocated to each of these channels. The distribution of subblocks is stored in the additional memory unit – block_table.

The array can be used for a simulation of a buffer memory. Support of dynamic reallocation of buffer space at the system level can be implemented in same way as for RTL model. However this implementation would significantly increase the simulation time. It only makes sense in cases where it is necessary to simulate and verify the reallocation mechanism.

Otherwise, for each virtual channel memory, which amount is equal to the total size of buffer space, can be allocated. The amount of memory that will use in each array is specified when configuring.

The methods of reading and writing data from these arrays must support the number of simultaneous reads and writes corresponding to the possibilities of the block memory in RTL model.

A disadvantage of the proposed approach is that redundant memory is allocated to each virtual channel. However, it is compensated by the lack of the block_table and simplified memory management scheme that reduces the simulation time.

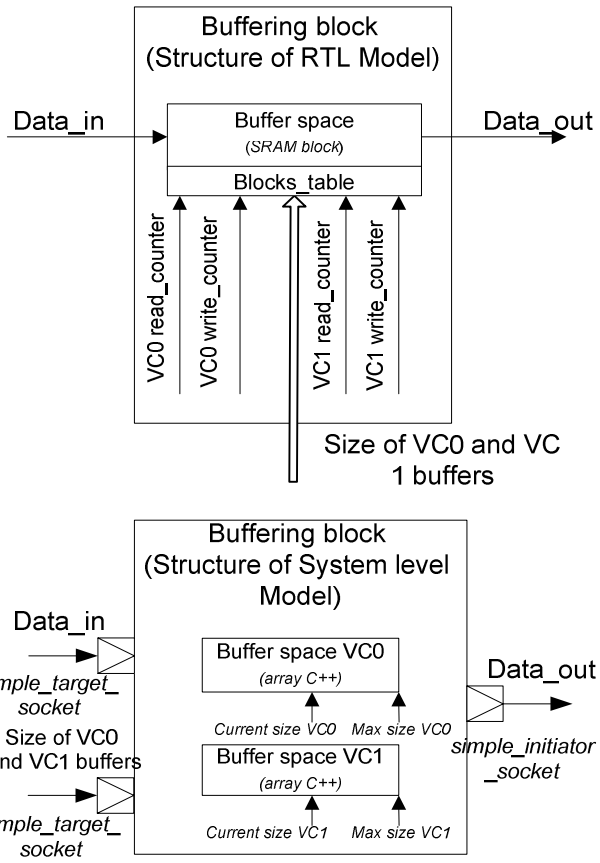


Fig. 6. The example of reconfigurable buffering block implementation

Standard features of SystemC are enough to support this reconfiguration type.

As it is shown above, the dynamic reconfiguration of a component can be implemented with using of the reconfigurable library elements (one element can be configured as NAND, NOR etc.). In order to support this type of system-level reconfiguration two approaches can be used.

The first of these “low-level” – each type of reconfigurable element is associated with a class, Fig. 7. The behavior of a specific instance of a class can be dynamically configured. It can be implemented with using of operator case (C++). Such approach does not lead to additional overheads (increase of simulation time) for each separate component.

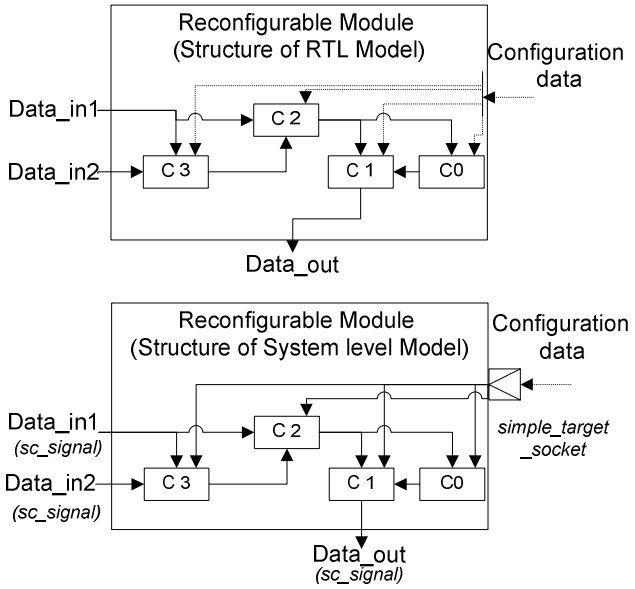


Fig. 7. The example of module based on dynamically reconfigurable library elements

However, the specification of reconfigurable block can include a large number of such elements. This may lead to increase in simulation time. In addition, the specification will look “difficult to read”.

In the second approach, the reconfiguration possibilities are specified not on library elements layer but on the layer of logic functions.

This kind of specification will not be “difficult to read”. However, the number of functions that can be implemented using a configurable set of components may be very large.

Therefore, for the implementation of this approach, it is advisable to use a dynamic child process with the required functionality using *sc_spawn*.

Using of this approach will lead to an essential increase in simulation time. It should be used when the number of configurable elements is several hundred or more.

Example of system level model with the ability to configure is presented on Fig. 8. Instance of the class “Manager of reconfiguration” controls process of system reconfiguration. TLM-2.0 socket classes may be used for a data transmission configuration between “Manager of reconfiguration” and other system modules. For example, the look-up table contains information about reconfigurable finite state automate. We can download/update look-up table using the TLM-2.0 sockets. Also using of the TLM-2.0 sockets is beneficial for working parameters setting inside the module B and for data communication between the module A and the module B, the module B and the module C or the module D. The SystemC events Event1 and Event2 depend on value of reconfiguration field. As example, when value of reconfiguration field is equal to 1, then Event1 is notified. When value of reconfiguration field is equal to 2, then Event2 is notified. The process F1 is sensitive to the Event1. The process F2 is sensitive to the Event2. It allows to run different processing functions over received data according to reconfiguration parameters. Using *multi_passthrought_initiator_socket* helps to transmit data into module C or into module D.

The data flow corresponding to the first configuration is represented by the dotted line on Fig. 8. The data flow corresponding to the second configuration is represented by the dashed line.

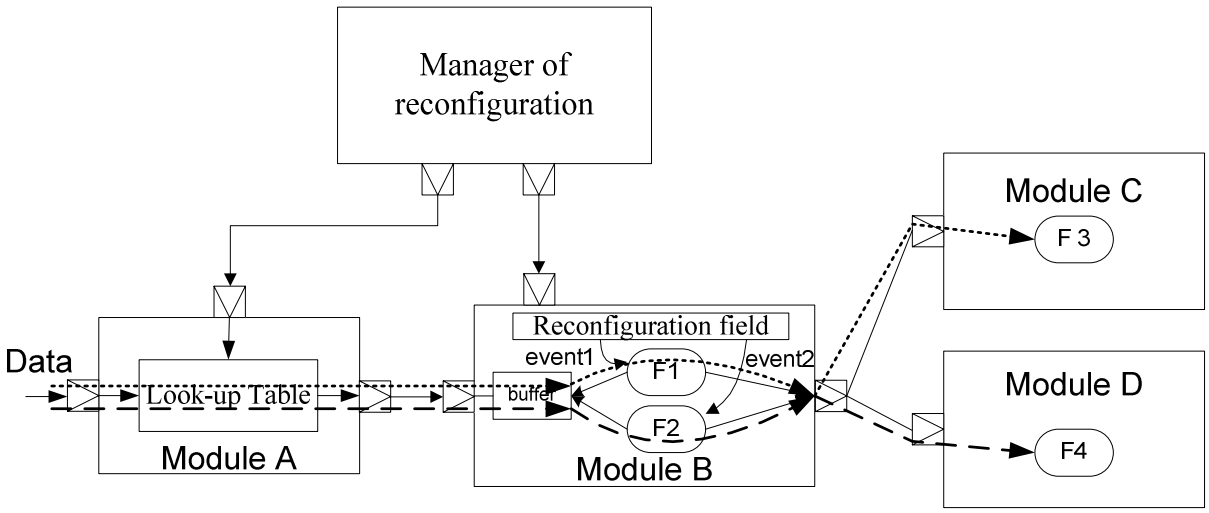


Fig. 8. Example of system level model with the ability to configure SoC based on the ASIC technology

## IV. CONCLUSION

We consider different approaches for development of dynamic reconfigurable SoC implemented on the ASIC and FPGA technologies. Requirements for system level module of SoC are described in details. Also we review existing techniques for the development of the system level models implemented on FPGA. Furthermore, we offered approaches for the development of the system level models of dynamic reconfigurable SoC implemented on ASIC. In the part 3 we present the information about the simulation of the dynamically reconfigured interconnection structure between components and simulation of dynamically reconfigured functional blocks.

Also, in this paper we consider features SystemC and TLM for tasks of dynamic reconfigurable SoC modeling. We expand advantages and disadvantages using the latest version of the SystemC standard in the system level modeling area.

## ACKNOWLEDGMENT

The research leading to these results has received financial support from the Ministry of Education and Science of the Russian Federation according to the base part of the state funding assignment in 2015 project no. 1810 and under grant agreement no. RFMEFI57814X0022.

## REFERENCES

[1] Nikolaos S. Voros, Konstantinos Masselos *System Level Design of Reconfigurable Systems-on-Chip.* Springer, 2005.

[2] S. Khawam, I. Nousias, M. Milward, Y. Ying, M. Muir, and T. Arslan,"The reconfigurable instruction cell array," *IEEE Transactions on Very Large Scale Integration Systems*, vol.16, no.1, Jan 2008, pp. 75–85.

[3] Stensgaard, M.B. ReNoC: A Network-on-Chip Architecture with Reconfigurable Topology, *in Proceedings of Second ACM/IEEE International Symposium*, 7-10 April 2008, pp.55-64.

[4] Yana E. Krasteva, Eduardo de la Torre, Teresa Riesgo, "Reconfigurable Networks on Chip: DRNoC architecture", *Journal of Systems Architecture*, vol.56, issue 7, July 2010, pp.293–302.

[5] NEC's Dynamically Reconfigurable Processor, Web: http://www.necel.com/drp/en/index.html.

[6] Ian O'Connor, Ilham Hassoune, David Navarro, "Fine-Grain Reconfigurable Logic Cells Based on Double-Gate MOSFETs", *IFIP Advances in Information and Communication Technology,* 2010, pp. 97–113.

[7] I. Hassoune, I. O'Connor, "Double-Gate MOSFET Based Reconfigurable Cells", *Electronics Letters,* vol.43, issue 23, Nov. 2007, pp.1273–1274.

[8] Richard Neil Pittman, "Partial Reconfiguration: A Simple Tutorial", *Technical Report*, Redmond, February 2012.

[9] Wang Lie, Wu Fengyan, "Dynamic partial reconfiguration in FPGAs". *Third International Symposium on Intelligent Information Technology Application*, Nov. 2009, pp. 445 - 448.

[10] Xilinx Inc Partial Reconfiguration Design with PlanAhead, Web: http://www.xilinx.com.

[11] Vincenzo Rana, David AtienzaMarco, Domenico Santambrogio, Donatella Sciuto, Giovanni De Micheli, "A Reconfigurable Network-on-Chip Architecturefor Optimal Multi-Processor SoC Communication", *VLSI-SoC: Design Methodologies for SoC and SiP, IFIP Advances in Information and Communication Technology*, vol. 313, 2010, pp. 232–250.

[12] IEEE 1666-2011 Standard for Standard SystemC Language Reference Manual

[13] Jeremy Bennett, "Building a Loosely Timed SoC Model with OSCI TLM 2.0", *Embecosm,* note 1, issue 2, May 2010.

[14] Kenji Asano, Junji Kitamichi, Kenichi Kuroda, "Dynamic Module Library for System Level Modeling and Simulation of Dynamically Reconfigurable Systems", *Journal of computers*, vol. 3, no. 2, Feb. 2008, pp.55-62.

[15] Andreas Raabe, Philipp A. Hartmann, Joachim K. Anlauf, "ReChannel: Describing and Simulating Reconfigurable Hardware in SystemC", *ACM Transactions on Design Automation of Electronic Systems*, vol. 13, no. 1, January 2007, pp.1-16.

[16] ReChannel - A Reconfiguration Simulation Library for SystemC, Web: http://www.ti.uni-bonn.de/static/research/ReChannel.

[17] Alisson Vasconcelos Brito, George Silveira and Elmar Uwe Kurt Melcher, "A Methodology for Modelling and Simulation of Dynamic and Partially Reconfigurable Systems", *Dynamic Modelling,* Jan. 2010, pp.29-48.

[18] F. Duhem, F. Muller, P. Lorenzini, "Methodology for Designing Partially Reconfigurable Systems Using Transaction-Level Modeling", *in Proc. of Conference on Design and Architectures for Signal and Image Processing (DASIP)*, Nov. 2011, pp. 1 – 7.