High-Level System-on-Chip Simulator

Aleksandr Orlov, Alexey Syschikov
State University of Aerospace Instrumentation
Saint-Petersburg, Russia
alexanderorlov@live.ru, alexey.syschikov@guap.ru

Abstract

A high-level simulation is a significant part of the multicore system-on-chip (SoC) software development process. It allows executing programs and performing functional debugging on the high-level model of the SoC without going into details of SoC heterogeneous cores: a specific command set, a processes interaction, a communication system specifics etc. The high-level simulation is also a necessary part of the SW/HW co-design tool flows.

This paper presents the developed high-level SoC simulator. This simulator allows executing coarse-grain programs in the configurable SoC-style distributed environment with heterogeneous processing elements and an interconnection. A parallel data-processing workload (SoC program) is been defining as a scheme of interacting processes with a C/C++ implementation and specific characteristics. Various simulation statistics allows investigating characteristics of a developed program (maximal parallelism levels, computation space requirements, amounts of interaction data), abilities of SoC architecture to perform such workload (processing elements and communication system occupation, buffers size distribution, queues etc.) and characteristics of program execution (processing time, latencies, constraints).

Index Terms: systems on chip, SoC, high-level simulator.

I. INTRODUCTION

The microelectronics market constantly brings new and stringent requirements for appearing products. A consumer wants to get high-speed, reliable and at the same time small-sized and low-power products. These contradictory requirements are compounded by the fact that microelectronic products are rapidly aging. Therefore a special attention should be paid to a permanent reduction of a time to market for new products.

A simultaneous development of hardware and software components of a new product is a way to solve the task of a time to market reduction.

Systems on chip (SoC) are the modern and overwhelming trend in a computing systems development. A SoC is an integrated circuit that integrates all computing components of a whole computing system or another electronic system into a single chip.

New versions of systems with traditional architectures do not have significant differences from the previous ones, so a software development can be done on the previous versions. In contrast, SoCs of different generations or even inside a single generation are significantly different in all aspects, including a set of processing elements their types, quantity of existing processing elements and their characteristics, a communication system and its properties. So SoC software cannot be executed and tested before the existing architecture will be implemented. SoC simulators are used to allow software developers producing software without an access to the implemented hardware or simultaneously with SoC hardware development.

With the financial support of the Ministry of Education and Science of the Russian Federation

SoC programs can be tested using simulators. A simulator is a program which helps rapidly and with minimal cost to inspect, set and test real system program-logic model without its building and provides to a user an opportunity to invoke and debug program for this model on it.

SoC simulators that are developed by SoC producers are low-level simulators, which implement a platform down to details of processing elements specifications with an instruction system, registers, memories, communication systems etc. It's useful for final software approbation, but significantly slows down the program design on the level of coarse-grained implantation. SoC simulators with high level of abstraction allow executing programs not being absorbed in SoC details such as processor commands, different internal hardware delays, elements architecture etc.

II. MAIN PART

A. Work purposes

Our task is to provide the configurable SoC-style environment that allows executing SoC programs, which are defined as a network of interconnecting operators. A SoC program is written using the VPL visual programming language [1, 2]. The SoC has the specific configuration which is presented in special platform description file.

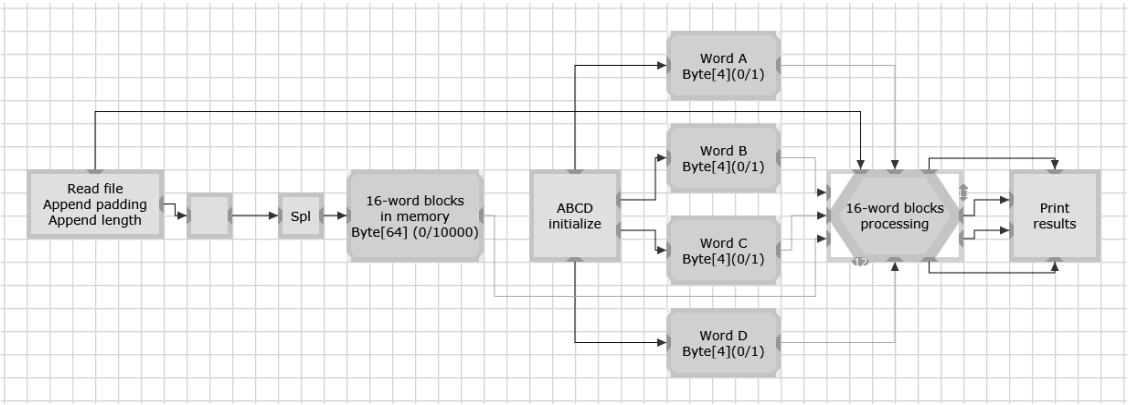

Fig. 1. Example of the SoC program on the VPL language

The purpose is to create the high-level system-on-chip simulator which will allow invoke programs intended for such systems and not to go deeper into the hardware description, such as processing elements command set, architectures elements, different electric signals delays, etc.

This simulator should:

- Provide an ability to model the systems-on-chip with an arbitrary configuration (it can consist of routers, channels, busses, processing elements of different types and characteristics);
- Model the next time characteristics:
  - time delays of data packets sending and reception for each SoC device;
  - time delays during data packets transmission through channels;
  - time delays during program processes invoking on processing elements;
  - time delays for accessing common memory etc..
- Collect statistical information about a simulation process;

- Show textual and graphical information after a simulation process is finished.

B. Simulator description

Idea of a simulator creation is not a something novel. There are different abstraction level simulators from functional simulators of a whole system level down to simulators of system commands level and simulators, which can simulate an accurate hardware structure on functional blocks level.

The simulation engine that was selected for the high-level SoC simulator development is the DCNSimulator project (Digital Communication Network Simulator) [3]. It is the cross-platform simulation engine which is written on the C++ language. The simulation core is based on the open source SystemC modeling library. The Qt library is also taken as a part of this simulation engine. It provides tools for working with files, graphics, different data containers and etc. The main advantages of Qt library is the wide set of provided tools, a high abstraction level and the good documentation. Qwt (Qt Widget Toolkit) is used to display the simulation results in the form of diagrams.

The SoC configuration is represented in the special file (Fig. 2). On the current state of implementation only directly connected and network on chip (NoC) subtypes of the whole SoC class is implemented in the simulator. An implementation of other communication systems is in progress. So, the SoC configuration consists of channels, processing elements and routers (optional).

```
<?xml version="1.0" encoding="UTF-8"?>
<platform>
  <devices>
    <pe name="PE1"></pe>
    <pe name="PE2"></pe>
    <pe name="PE3"></pe>
    <router name="Router1"></router>
  </devices>
  <channels>
    <channel src="Router1" dst="PE1" ></channel>
    <channel src="Router1" dst="PE2" ></channel>
    <channel src="Router1" dst="PE3"></channel>
    <channel src="PE1" dst="Router1" ></channel>
    <channel src="PE2" dst="Router1" ></channel>
    <channel src="PE3" dst="Router1"></channel>
  </channels>
</platform>
```

Fig. 2. Sample file with SoC platform configuration

The channel is intended for a connection of different hardware devices: a processing element with a router or directly with another processing element. It can simulate a time delay of packet transfer through it.

The PE is a multi-allocation single-task computing element. Multiple tasks can be allocated to a single PE, however only one task can be executed in single moment on an exact PE. Moreover, no task switching is allowed, i.e. once started the task occupies a PE until it finishes. All allocated tasks can receive input data to its buffers independently of occupation of computation recourse. Tasks readiness conditions are checked on every data arrival: ready task is either put to execution (if a PE is free) or put to the ready tasks queue (if PE is busy). The PE can simulate its occupation, i.e. a time delay of a task invoking. The PE also has input and output data buffers. After data packets arrive to

input buffer they are parsed (a destination task and an input port is extracted from a data packet) and data from them are sent to a specified task input port. An output buffer is intended for storing output data packets while a channel is occupied and a PE cannot send them through a channel.

The router is a network device that routes data packets through an interconnection network. Each router has as many ports as many channels are connected to it. Each router has a routing table with processes (tasks) allocation information. After a router receives data packet it reads information about a packet destination and in accordance with a routing table sends this packet to an appropriate output port.

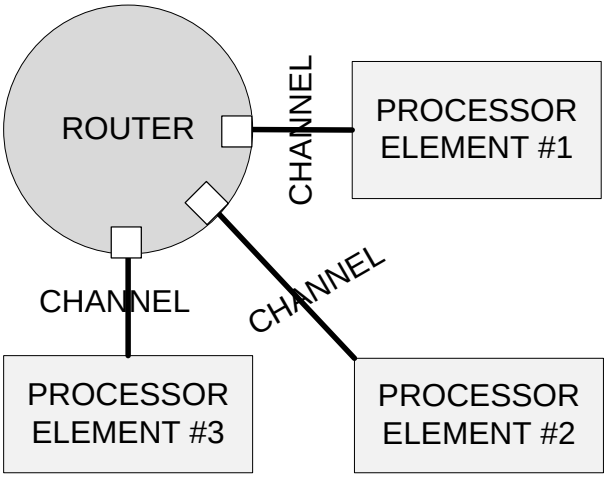

Fig. 3. Sample system-on-chip presented on Fig.2

As it was said before, a program for SoC is written on the visual scheme-oriented programming language VPL. There are the next scheme object types in programs on the VPL language:

- Operators;
- Data;
- Links;
- Structures (subprograms);
- References.

Operators are objects which implement data processing functions. These functions are written using the special API on the C or C++ programming languages.

Data objects are passive partially shared data storage. Only operators that have links with data object can access it. An access type for every operator is restricted by the link type: it can have “read”, “read-erase” and “write” access. The VPL language model does not specify implementation details for data objects, only the rules of functioning. So, in the presented simulator the next data object interaction routines is implemented: if an operator wants to work with a data object it needs firstly to send a request to get an access to data from data storage or to get a confirmation to write to it.

Links are intended for describing connections of program objects with each other.

Structures are mechanisms of a hierarchical program structuring and keep a subprogram description inside them. Subprograms has no significant difference from a main program, the only difference is that subprograms has references objects which

connect them with the objects of a parent program part. Each reference keeps a numeric value that specifies a structure port number to which it refers. The reference with an input link (Fig. 4a) shows that data which was sent through this link will be put on an output structure port. The reference with an output link (Fig. 4b) shows that data will be got from an input structure port.

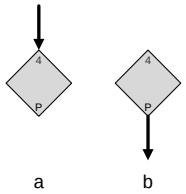

Fig. 4. Reference examples (a – with input link, b – with output link)

C. Simulation preprocessing

After files with a program and a SoC description are got, the simulator prepares for a simulation. At first, the simulator parses a program file and gathers information about objects which will take part in a simulation. After that the simulator creates processes for every program object that will take part in a simulation. Different process types implements different types of program objects. The simulator parses a configuration file with a SoC platform description, creates simulation objects for all hardware devices which is presented in this file and connects them with each other.

On the next step it is necessary to evenly allocate processes on PEs. The simulator can use an external allocation information or use the embedded round-robin algorithm [4]. It is the load-balancing algorithm for distributed computing systems by sorting and ordering of its elements on a circular loop.

The next step is building of routing tables for routers according to platform configuration and an allocation. Routing tables have routes between processes which take part in a simulation. Routes between them are constructed using the Dijkstra's algorithm [5]. It helps to find a route between two objects with a minimal cost.

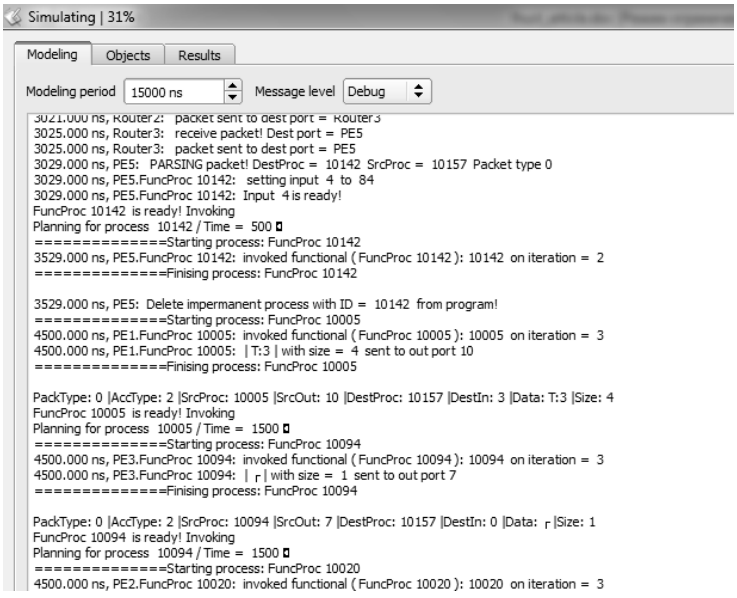

Fig. 5. Simulator tool's window

D. Simulation process

After preprocessing the simulation process is ready to be started. It is necessary to press the "Start" button on the tool window to start a simulation. Before pressing the button user can set a simulation time and change a level of debugging information output (level with all debugging information, level only with information about warnings and level with information about critical errors).

When a simulation is running, debugging information is shown in the tool window (Fig. 5). This information can include processes time invoking, processes readiness, packets movement in network, etc.

After a simulation finishes a user can explore statistical information about a program simulation and SoC parameters. Information is presented in textual and graphical forms.

Textual information contains files with information about each PE, each process and general information about simulated PEs and simulated processes. For example, a text file with general information about simulated processes contains information about:

- Time when process started and finished invoking;
- Name of process;
- Processes ID;
- PE name on which the process was invoked;
- Number of processes iteration.

```
1500 ns: FuncProc 10005  is invoking on PE1  on iteration  1
1500 ns: FuncProc 10094  is invoking on PE3  on iteration  1
1500 ns: FuncProc 10020  is invoking on PE2  on iteration  1
1521 ns: IfProc 10157  is invoking on PE4  on iteration  1
2033 ns: FuncProc 10142  is invoking on PE5  on iteration  1
3000 ns: FuncProc 10020  is invoking on PE2  on iteration  2
3000 ns: FuncProc 10094  is invoking on PE3  on iteration  2
3000 ns: FuncProc 10005  is invoking on PE1  on iteration  2
3017 ns: IfProc 10157  is invoking on PE4  on iteration  2
```

Fig. 6. Text file with general information about simulated processes

Graphical information contains graphics of processor elements occupation in time, graphics of processes invoking in time, graphics with information about ready processes queue size on processor elements, buffers size distribution etc. For example, the Fig. 7 has a diagram with a statistic of PE occupation. The axis X represents time and the axis Y represents 2 states of PE occupation – 1 for PE is occupied and 0 for PE is free.

III. CONCLUSION

The high-level SoC simulator which is presented in this paper is intended for simulation and debugging of different computations on a SoC model before it is implemented in a hardware. The high-level SoC simulator provides a lot of useful information about simulated platform.

The future work on this project will introduce new features of the simulator. It will be new different statistic information, new types of hardware object and other improvements.

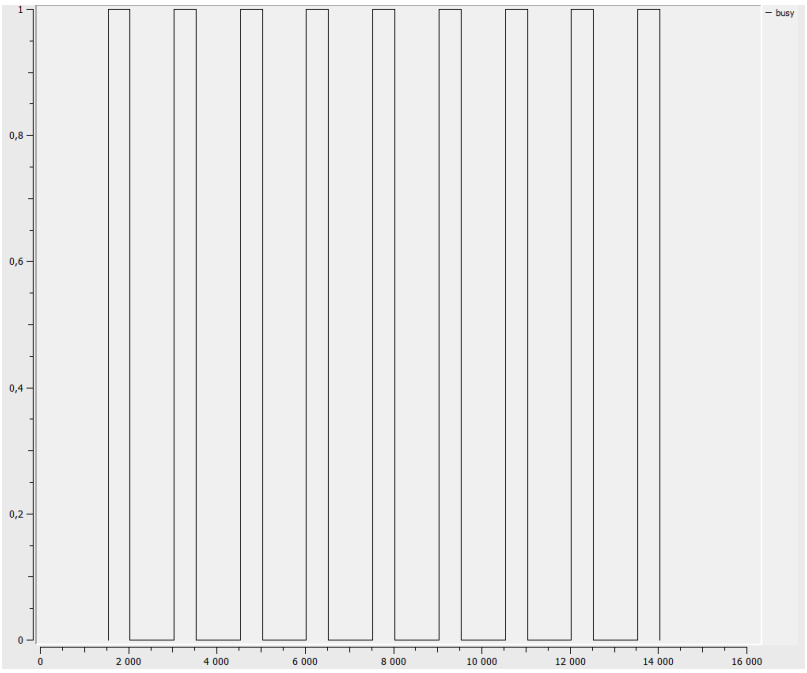

Fig. 7. Graphic of processor element occypation

REFERENCES

[1] Syschikov A. Programming technology for heterogeneous systems on chip. SUAI scientific session, proceedings. Saint-Petersburg, 2008, c.133-139.

[2] Ivanov V., Sheynin Y, Syschikov A. Programming model for coarse-grained distributed heterogeneous architecture. Proceedings of XI symposium on redundancy in information systems. Saint-Petersburg, 2007, c.246-250.

[3] Sheynin Y.E., Volkov P.L., Onischenko L.V., Razhivin D.B., Cherniy A.S., Eganyan A.V., Nikolsky V.F., Kosyrev S.A., “Software support of the VLSI family “Multicore-designer” for the construction of the parallel structures and distributed signal processing systems”, “Questions of Radio electronics”, a series of “Electronic computing equipment (EWT)”, issue 3, 2008.

[4] Round-robin. Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Round-robin.

[5] Dijkstra's algorithm. Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm.