SystemC Modelling of the Embedded Networks

Valentin Olenev, Yuriy Sheynin, Elena Suvorova

St-Petersburg University of Aerospace Instrumentation
190000, St-Petersburg, Bolshaya Morskaya 67, Russia
Valentin.Olenev@guap.ru, Sheynin@aanet.ru,
suvorova@aanet.ru

Sergey Balandin, Michel Gillet

Nokia Research Center and Nokia Devices
Itamerenkatu 11-13, 00180 Helsinki , Finland
Sergey.Balandin@nokia.com.
Michel.Gillet@Nokia.com

Abstract

Following to the rapid development and broad acceptance of computers and microprocessors, communication standards and network technologies development engineers are compelled to equally increase complexity of development, manufacturing and testing of new products. Every year complexity of the embedded systems is increasing exponentially and this is true not only for the HW components, but also for embedded software. These forced the system developers to look for the new means that would allow them to cope with increasing projects complexity, to reach competitive productivity and guarantee that the developed systems satisfies customers' expectations.

Thus modelling takes more important role in the development process as a solution to perform detailed check of the specification and project verification to a stage of physical realization. Modelling could be applied to the different stages of system design and a number of different languages and modeling platforms can be used. The models could have different structures and internal logic.

This article describes a role of modeling inside the systems development course, discuss the advantages and tricky points that the developer can face while using modeling approach. Besides, the article describes SystemC modelling language, which is a good solution for making models of the embedded networks.

Index Terms: Modelling, SystemC, Embedded systems.

I. Introduction

The embedded system is a special kind of computer system where the processing logic is built into the device which it operates. Such systems are developed to carry out one or several specialized functions and usually have strict real time computing restrictions. The most standard set of requirements to the key features of such systems are:

- Small power consumption;
- Small physical sizes;
- Absence of active cooling, i.e. the processor is not cooled or the small radiator can be used;
- The processor, system logic and also some other schemes, are often combined on one chip.

The embedded systems can be implemented for the single-chip microcontrollers, specialized or universal microprocessors. Examples of such embedded systems are cash dispenses, payment terminals, handheld computers, telecommunication equipment, similar systems are built in even such devices, as traffic lights [3].

The embedded systems are managed by a processing kernel which is either the microdispatcher or the digital signal processor (DSP). It is also important to remember that all embedded system are targeted in solving a certain set of tasks and while making the system optimization engineers can either focus on reducing the size and cost, or concentrate on increasing reliability and functionality. The right balance here is another important subject to be addressed [1].

II. MAIN PART

A. Embedded systems design flow

Analysis of design development prospects at various levels of abstraction (fig. 1) shows that whilst in 1990 a project realization (from the logic level downward) took 90% of all volume of design works in 2000 this share was reduced to 55%. And by 2010 designing at architectural and functional levels will make 70% of the project design efforts, with only 30 % necessary for the project realization in the chosen element basis.

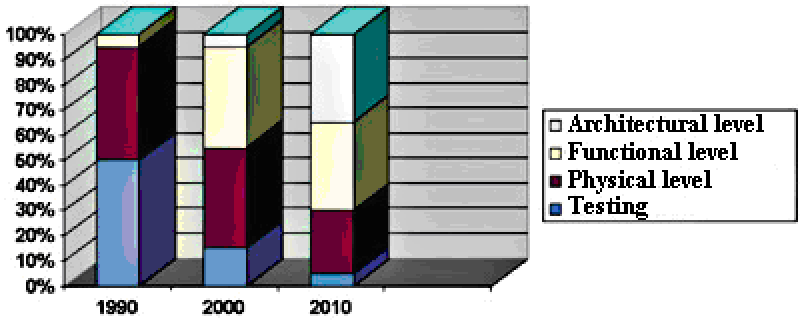

Figure 1. Designing tendencies at various levels of abstraction

Nowadays systems design encounters a number of difficulties caused by increasing complexity of projects, increasing requirements to products reliability, power consumption and necessity of the shortest terms of the project design. The traditional route of designing "top-down" does not allow meeting these requirements. In modern systems design, containing one or several processor kernels, the so-called "spiral" route is used. This route is shown on fig. 2 and includes the stages:

- **Conceptual system design;** the primary goal of the stage is development directions choice, analysis of the system in design and development of its specification drafts;
- **Specification;** this stage is to get the final version of specification, and its model in a high level language (usually in C/C ++, SystemC, SDL);
- **Logical design**; converting of the executable project specification to the register transfer level (the result is a specification in Verilog/VHDL) and further to the gate level;
- **Project verification;** verification of the project and design decisions on conformity to the initial specification and to other requirements, in the course of design and detailed elaboration process;
- **Physical design;** this stage begins from a technology and library basis choice and finishes on the final specification of the project in the GDSII format.

In the process of embedded systems design it is the conceptual level that is critical for estimation of general system characteristics. At this level the first drafts of the specification of the system is created, it allows to investigate and estimate various design decisions, to choose the optimal decision which of them to implement further [4].

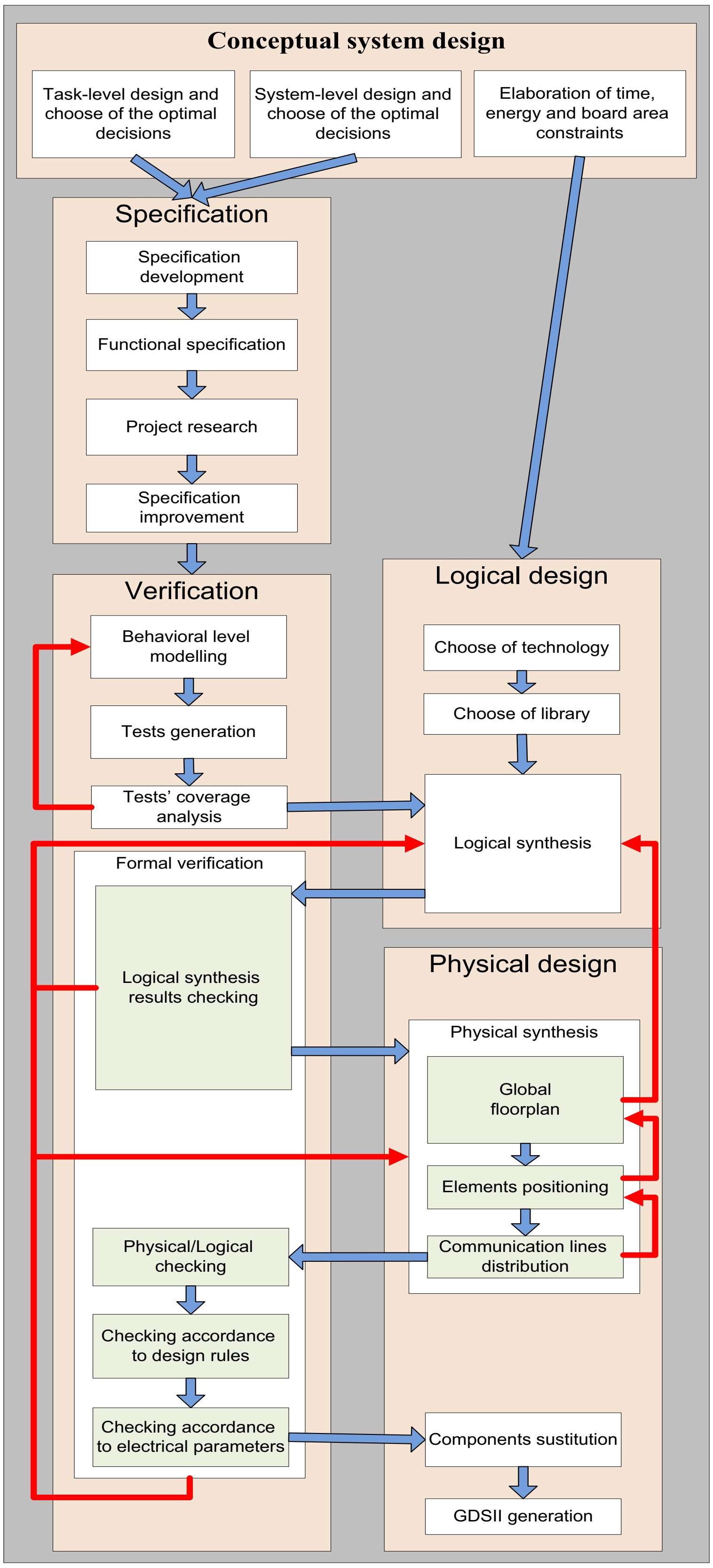
Conceptual system design
Task-level design and choose of the optimal decisions
System-level design and choose of the optimal decisions
Elaboration of time, energy and board area constraints
Specification
Specification development
Functional specification
Project research
Specification improvement
Verification
Behavioral level modelling
Tests generation
Tests' coverage analysis
Formal verification
Logical synthesis results checking
Physical/Logical checking
Checking accordance to design rules
Checking accordance to electrical parameters
Logical design
Choose of technology
Choose of library
Logical synthesis
Physical design
Physical synthesis
Global floorplan
Elements positioning
Communication lines distribution
Components sustitution
GDSII generation

Figure 2. General design flow

B. Modelling for the specification development process

Main objective in the course of the project specification are definition and specification of basic functions of the system and creation of an executable system model. This system model is used to verify the correctness of system operation from the functional point of view, and also to estimate necessary hardware resources and define system architecture. It is used also to check the mechanisms and approaches in the specification for correctness and compatibility.

At this stage following problems are solved:

- Creation of the system functional model, description of the system in terms of the algorithms and functions which it should implement, without binding to further implementation techniques;
- System modeling in its operational environment with real data and signals;
- Specification of the system architecture in terms of necessary resources and hardware-software realization of functional model.

With the executable system specification, behavioral models and the general architecture, some stages of the further design, verification and topological implementation of the system could run in parallel.

The general specification development flow at this stage is shown at the fig. 3.

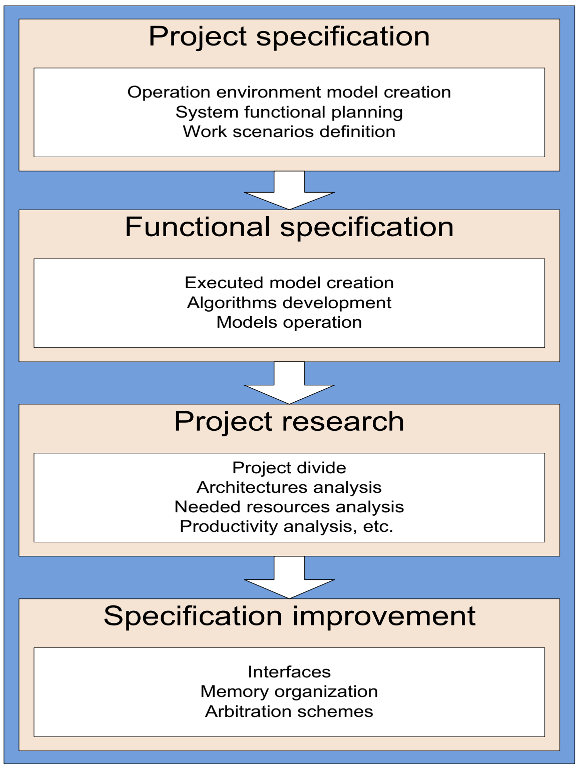

Figure 3. Specification development flow

At this stage various electronic drives, control systems, relative positioning and movement of objects can be modeled.

Then the functional specification of the system is created. Its purpose is to define and model the system functionality in terms of its operation algorithms. System behavior as a whole or its separate blocks could be set and modeled here. At this level of system functionality used to be modeled with real data and signals.

In the phase of project analysis, modeled functions are transformed and separated to perform them on a number of platforms, or architectures which contain various sets of components, such as programmed processors, memory, ASIC, or systems-on-chip. Various kinds of estimations are used here to find the optimal architecture which fits with set criteria, such as real time operation, productivity, cost, power consumption, etc. Software functions could be estimated in terms of the code size and the worst execution time, performance measured by processor operation cycles; hardware functions – by a number of equivalent gates.

Finally, system specification enhancement is done; more detailed description of system architecture is made to pass it to the Design stage. Such system level description can contain some details of the subsequent realization, but the functional part consists of behavioural models on C/C ++/SystemC languages. Then joint hardware-software design runs using concrete processors models and buses (functional models), the blocks described on design languages of equipment [4].

C. Modelling for systems implementation validation

Functional verification is more and more important in the design process. Formerly design was understood as project development at register transfer level and verification was made by logical modeling tools. At present verification starts at the behavioral level, at the stage of the common project specification development.

Main requirements for structure of functional design and verification tools are:

- analysis of the architecture, productivity and other system parameters;
- hardware-software design of systems, possibility of joint development and verification of hardware and embedded software;
- system design with processor blocks, use of processors' models in hardware and software development;
- uniform design environment, from the system level to the register transfer level and the gate level with C, C ++, SystemC languages support and hardware description languages such as VHDL and Verilog;
- libraries and high level constructs for functional blocks and communication channels, connectivity tables included;
- tools for the projects data control and documentation.

On upper levels of representation C/C ++ or SystemC languages are used. For modeling with C/C++ code some built in modeling kernel is used. It does planning and running a model according to the structure and behavioral functions of the project together with operating systems. When C/C ++/SystemC units are detailed step-by-step with transition to lower layers of abstraction, a developer can substitute C unit by a representation in HDL (VHDL or Verilog). Some EDA can also synthesize a block implementation version if it is represented at the C level in accordance with certain rules [4].

D. Modeling for device testing

The important stage in embedded systems design flow is testing of real systems on boards with an exact computer model of the standard according to its specification. Modeling is used for validation of a standard specification itself, for algorithms and systems testing.

Inputs/outputs of a written in one of simulation languages model, which runs in the computer, by specific interface units are connected to inputs/outputs of the real board – unit under test (UUT). Thus, developers can set various operating modes of the model by software and test response of the attached device and its work in point-to-point connections or in network structures. Thus with a model of the used interface or network standard a testing environment for testing, validation and certification can be feasibly built. Though there is no standard for building such testing environments logical approaches are pretty similar. An example of the Tester architecture is shown at the fig. 4:

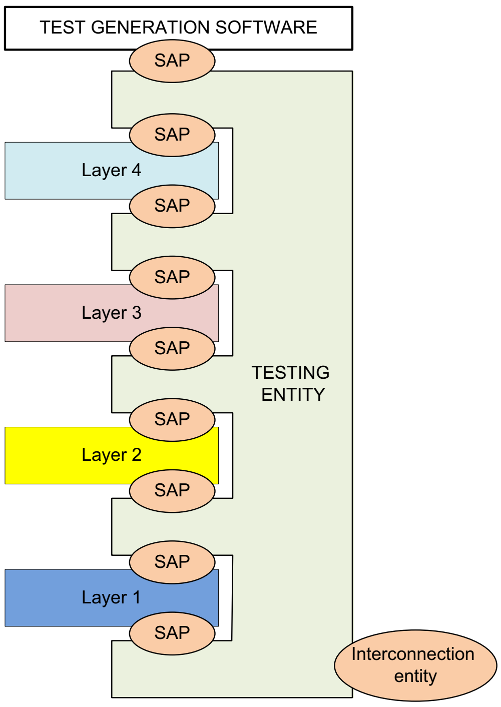

Figure 4. Generalized architecture of the Tester

A 4-layer protocol is taken as an example. Between each pair of its layers an additional inter-layer part of the tester entity is specified. Inter-layer can control everything that goes between upper layer and lower layer, can independently send data directly into layers, carry on event log, or simply pass data through itself. It can generate data and dispatch it directly to any of the layers; erroneous data also could be generated for testing.

At the highest level the test generation software is located. By it means one can generate tests for the underlying model using some customary programming language. Data passed through the model goes from Tester into the Interconnection Entity and through it to a UUT – a device prototype for testing, a device for verification, etc. We can also connect by interconnection entities a couple of testers (or more, e.g. for standards with multipoint nodes) and use such an installation for validation of the specification itself and its model. Tester could contain mechanisms for detection of industrial defects on the board also.

So, such a Tester allows:

- to validate the standard specification itself, e.g. in the course of the standard development and evolution;
- to validate the model itself for correspondence to the specification;
- to test prototypes or boards in production;
- to certify products, to verify products for conformance with the standard.

To program the Tester: based on the model Testing Entity, Interconnection Entity, Test Generator, a flexible programming language or a combination of languages for modeling are needed [2, 9].

E. Requirements to the design language

For system level design and verification at such kind of language needed that supports system level design abstractions and can deal with complexity of modern embedded systems. They should:

- should be based on a uniform notation for a possibility of simultaneous work by both programmers and hardware designers;
- support special primitives for verification at different abstraction levels, should be applicable to use at all levels. ;
- support software reuse, reuse some its structures, model components;
- support detail of implementation or synthesis up to the RTL;
- be able to integrate both hardware and software components models;
- processor simulators to execute compiled for them object codes;
- support integration of different computing models;
- allow co-simulation and interaction of models at different abstraction levels: functional, architectural, transactions level, PHY level;
- support modeling of communications at different abstraction levels and integration of these models into efficient system model [4].

F. SystemC modelling language

SystemC modeling becomes one of the most efficient and widely used method in studying, analysis or construction of multi-component systems, such as stacks of protocols, networks with a large number of nodes, or systems-on-chip, networks-on-chip, etc.

SystemC is a C ++ library, which is used for modeling of parallel systems. The library allows to describe multi-component systems and program components and to model their operation. Using events it models distributed in time operation of a modeled system.

The SystemC kernel defines data types, such as bit, vector types and comma-fixed types. Also there are kernel units, such as processes, events and channels. For implementation of mechanisms for interaction between parallel objects, elementary channels as signals or stack FIFO are created [6].

SystemC qualifies as a language for the specification, modelling, design and verification of systems. In these fields the SystemC language has a number of advantages:

- SystemC is based on C++; it is constructed on the basis of libraries extension and the simulation kernel, written in C++. Therefore, it is feasible to integrate all types of based on C++ models, including instruction set simulators, software models, hardware models, process and analyze modeling results at high level, with C or C++ code.
- SystemC uses such primitives as channels, interfaces and methods; it gives high flexibility in modeling that could be based on various computation models, provides possibility to integrate and use these models in parallel. There are system models

examples in SystemC that combine digital hardware models, an analogue blocks models, data flows, a software part and finite state machines.

- SystemC supports models at all abstraction levels and uses, within the scope of OSCI (Open SystemC Initiative), standard semantics at transactions level and API; it guarantees possibility to work simultaneously with different abstraction level models.
- Most of developers, software and hardware experts are familiar with C/C++ and it point makes cooperation in HW/SW design easier.
- SystemC supports modern methodologies of verification by means of SystemC multilevel verification library (SCV - SystemC Verification library). SCV supports creation of test platforms, writing transactions (entry signals, monitors, responses check), randomization of variables, imposing of constraints/links, as well as application of new methodologies of transaction based verification.
- SystemC has been created for support IP-blocks of based design, reuse of design and verification components, based on C++ and OSCI components interaction standards at transactions level.
- SystemC supports hardware modeling and detailing of a project to the RTL level. SystemC provides coherence of hardware, signal and bit-level components, supports both hand-held detailing and synthesis methods.
- Modeling of on chip communications is naturally supported in SystemC. SystemC provides channels, units, interfaces and methods which give flexible, convenient and efficient modeling of all sorts on chip communications - standard buses, point-to-point and networks [5, 7].

G. SystemC in system modeling

In a system design flow main modeling abstraction levels are:

- Functional modeling of system algorithms.
- Modeling of system architectures at the transactions level.
- Modeling of RTL and binding SystemC with implementation paths in the design flow.

SystemC is the language which is ideally suited for embedded systems modeling. It is used not only to describe models, but also to develop a test environment for these models and to use it for testing of real boards. Recent SystemC additions for verification, SystemC verification library [6], support application of SystemC in design validation and verification tasks.

However SystemC usage in writing test environment software is limited, as SystemC is not intended for an applications programming. Though new version 3.0 will contain possibilities for modeling and planning of programs, SystemC in itself would not become a software engineering platform, [6]. For such tasks SystemC could be used coupled with its parent language C++. SystemC could be used for modeling of application layer protocols, e.g. protocols for camera, display, memory remote access protocols and so on. Such kind of models, are specified in SystemC with possible use of pure C ++ for description of the applications themselves.

SystemC is often used also for testing of operation of several protocols, working one "over" another in a protocol stack. In this case several models should be implemented: Application layer, protocol model, Data transfer protocol model, etc. Then interaction of these models should be implemented and operation of an application layer protocol over the data transfer protocol could be tested. By SystemC it is easy to describe structures, which are responsible for configuration of underlying model under needs of overlying, as well as to set various variants of configuration and functioning of the "upper" protocol. An example of such model architecture is shown on the fig. 5.

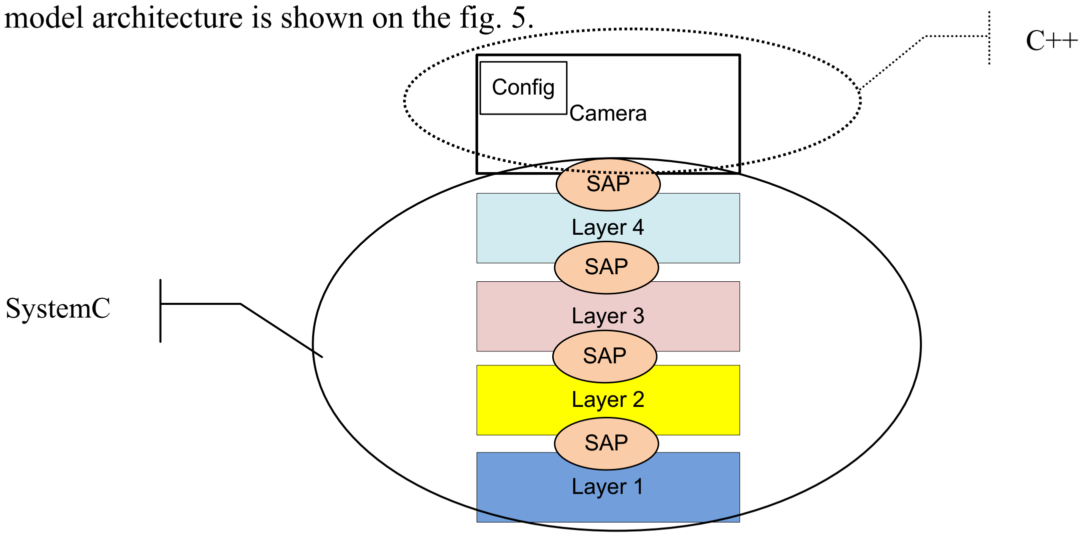

Figure 5. Example of the camera protocol operation over an abstract data transfer protocol

H. Verification with SystemC

SystemC verification library (SCV) offers many new possibilities for testing. It has been created as a library of extensions based on SystemC and SystemC libraries.

SCV helps to create verification IP-blocks, which could be considered like IP for design and like IP to reuse in other projects. Considering a large amount of efforts (during the design) for verification, feasible creation verification IP becomes very important.

General SCV library verification means are:

- Self-diagnostic (introspection) and possibility to work with any type of data items. Thus SCV could manipulate with user-defined data types and point to them – for example structures displaying packets and frames. For instance it gives PLI (Programming Language Interface – Verilog) a way to access such objects and to define all attributes or characteristics of these types.
- Transactions writing, that makes transactions-based verification (TBV) a supplementing part of transaction level modeling. Thus TBV, by marking event set as one “transaction”, allows to write simulation at the separate events level and to abstract result to common transactions of test platform; marking and writing transactions at this level of abstraction could be done. SCV allows to group linked transactions in streams; that gives a possibility to search in results of simulation, create high level verification transactions, observe and inspect programs.
- A simple randomization based on the set of limitations. Randomization is a powerful method in modern functional verification. SCV supports some types, including simple randomization. For instance, data addresses are generated by a simple interval function from a set of correct addresses. Simple randomization can create also distribution of values based on more complex sets of correct intervals, and to set empty intervals.
- Randomization allows to link probabilistic distribution with values or with a subset of values in defined interval. So complex distributions (for example, binomial, the Poisson, normal) can be associated with test platforms. It is also a good way to represent real statistical distributions in a test platform.

- Complex casual scenarios on test platforms could be generated by setting complex conditions and combinations in specification of allowable values sets of testing variables. For example, for communication applications it is possible to generate distribution of packages depending on addresses intervals, occurrence of errors, and some other constraints. SCV has a mechanism for defining constraints and finding decisions that satisfy to all constraints; decisions are distributed in regular intervals on all admissible values.
- SCV also provides a simple database to store and investigate verification results, gives simple SystemC to HDL simulator linkage mechanism; compatible mechanism for errors processing and debugging, error detection mechanism [5, 7, 8]

III. Conclusion

Modelling occupies very important role in embedded systems design process. It allows to accelerate and to simplify the design process for improvement of the standards quality and avoidance of specification errors. Also it helps to save the finance of the companies-developers. SystemC language occupies leading positions of modelling, as the most adapted, simple and clear language for off-site users. New SystemC standard updates also introduce ability to verification that expands the language applicability sphere. The above-stated analysis proves that using of SystemC for embedded systems modelling extremely justified and convenient both at usage pure SystemC and in a combination with other languages for solution of the various purposes on the trend.

SystemC combination with other simulation languages is very perspective direction. For example, there is a possibility to use SDL and C++. Thus, there is a possibility to plunge SystemC to SDL, which can help to avoid some SDL usage complexities. Global blocks, such as protocol layers and interaction of these blocks could be described in SDL, but internal mechanisms of a layer are more convenient to implement on SystemC. It allows using data types and mechanisms which are not present in SDL. Also for testing it is better and more convenient to write various channels on SystemC, using FIFO, inheritance of C ++ and clocks (clk) [10].

Thus, the modelling area develops actively, and engineers search new ways of production improvement, for operation simplification and the extension of modelling applicability.

Acknowledgment

The authors would like to thank Finnish-Russian University Cooperation in Telecommunications (FRUCT) program [12] for support of the related R&D activities, Nokia's and NSN university collaboration programs in Russia for supporting FRUCT community and all FRUCT experts for commenting and reviewing the paper.

References

[1] I. Shagurin "Systems-on-chips. Distinctive features of realization and perspectives, *"Electronics components #1"*, 2009.

[2] M. Gillet "Hardware/software co-simulation for conformance testing of embedded networks," *4th FRUCT seminar*, 2008, *http://www.fruct.org/images/contentmedia/S4_Hardware_software.pdf.*

[3] Wikipedia, free online encyclopedia, "Embedded Systems," *http://en.wikipedia.org/wiki/Embedded_system.*

[4] A. Bykhteev "Methods and facilities for systems-on-chip design," *ChipInfo*, 2008, *http://www.chipinfo.ru/literature/chipnews/200304/1.html.*

[5] V. Nemydrov, G. Martin, "Systems-on-chip. Design and evaluation problems," *Technosphera*, 2004.

[6] V. Olenev, L. Onishenko, A. Eganyan article “Connections in SystemC Models of Large Systems,” *from “SUAI scientific session” collection*, 2008.

[7] S. Swan “A Tutorial Introduction to the SystemC TLM Standard” presentation, 2003, *http://www-ti.informatik.uni-tuebingen.de/~systemc/Documents/Presentation-13-OSCI_2_swan.pdf* .

[8] J. Rose, S. Swan “SCV Randomization,” 2003, *http://www.openverificationfoundation.org/docs/scv_randomization.pdf* .

[9] E. Suvorova “A Methodology and the Tool for Testing SpaceWire Routing Switches,” 2009.

[10] A. Stepanov, V. Olenev, A. Rabin “Comparison of SDL and SystemC Languages applicability for the protocol stack modeling,” *from “SUAI scientific students’ conference” collection*, 2009.

[11] Official webpage of Finnish-Russian University Cooperation in Telecommunications (FRUCT) program, *http://www.fruct.org.*