Works in Progress in Embedded Computing

Using Architectural Abstractions in Embedded System Design

Vasiliy Pinkevich
Computer Science Department
ITMO University
Saint-Petersburg, Russia
vpinkevich@niuitmo.ru

Alexey Platunov
Computer Science Department
ITMO University
Saint-Petersburg, Russia
platunov@lmt.ifmo.ru

Abstract— **The conceptual part of complex embedded systems design includes the following key stages: system analysis of initial requirements, architectural and micro-architectural decisions generation, evaluation of decisions. During these stages, many important mechanisms of subsequent implementation are defined. These are the stages that are the least formalized and automated. The proposed method allows the design process to be partially formalized by the usage of computational mechanism concept as the central abstraction. The considered example regards to analysis of languages used together in complex embedded systems design with "immersion" to the level of custom system on a chip design. The comparison of design languages, carried out on the basis on the proposed approach, allows the design means for subtasks and subsystems to be chosen more effectively. The source code markup method is proposed as a tool for automated processing of multi-language projects targeted to work with design entities, which cannot be adequately and directly expressed by the standard languages means. In general, the demonstrated approach stimulates the designers to concentrate on "cross-cutting" conceptual mechanisms of a project and provides a way to monitor the adequacy of their multi-stage implementation.**

Keywords – embedded system; system level design; architectural abstraction; design space exploration; multi-language design.

I. INTRODUCTION

Embedded systems (ES) design process in its conceptual phase has to be based on methodologies of their complex representation [1]. In the literature, this level of consideration is typically attributed to electronic system-level (ESL) design [2, 3] or system-level design (SLD) fields, however, already in [4] it is noted that activities in this design stage are wider. We call these activities HLD – High Level Design. Abstract system concepts are used at the stage of ES high-level design. They largely form the ES project, but are not fixed in the ES implementation. This greatly complicates the control over the adequacy of system implementation in its "top-down" transition from one level to another within design process. Therefore, "cross-cutting" methods of working with conceptual information, that cover all ES design phases, are needed.

II. DISCUSSED PROBLEM

Languages for design, programming, modeling and other problems are critically important ES implementation instruments. They are actually platforms containing abstraction means to allow explicit allocation of conceptually important design units – classes, functions, macros, modules and other units. However, the serious problem with standard programming languages is that they do not allow system specification to be composed exactly in the same terms as designer thinks about it. The examples of entities, which are difficult to be expressed, include: cross-cutting mechanisms, with support scattered over the entire specification code (means of ensuring reliability, lower power consumption, etc.); mechanisms that affect multiple levels of a system, described in several languages (e.g. hardware description language and software programming language); any significant logical structures, unsupported by the language means. If developer has used abstractions of higher level than the language and standard library, these abstractions are typically left in his mind. Thus, it is necessary to have an opportunity to establish consistency between design abstractions, which are generated by the developer "in free mode", and constructions that are directly provided by the languages used in the design. For this purpose, both methodological framework and automation toolset are needed to allow the developer to use this approach in practice.

III. RELATED WORK

One of the known methods to solve the stated problem is the usage of domain-specific languages (DSL) and related tools [5]. However their usage may be limited due to the following reasons:

- initial project of the system is not formalized enough to be unambiguously realizable (synthesizable) from the specification;
- architectural information is unavailable so the system is considered only via its implementation;
- too much overhead for the creation or implementation of the language and tools that provide the required conceptual entities;
- legacy system support is required;
- manual optimization with the usage of low-level language is required.

This work was partially financially supported by Government of Russian Federation, Grant 074-U01.

There are examples of allocation and classification of conceptual elements for ES design and analysis in the literature. Typically, several maximally independent axes in the possible solution space is provided. Axes contain marks, which designate possible problem solutions. The marks may represent either various abstraction levels (see Fig. 1a, "design cube" – model for VHDL language [6]) or technical decisions (see Fig. 1b, "design space evolution" [7]). Furthermore, axis may represent the process that evolves in time within computer system life cycle (see Fig. 1c, "rugby model" [8]). Also, the known examples of this approach usage are VSIA taxonomy and ESL taxonomy [9], which consider the properties of ES structural units models. They can be applied for analysis of programming and design languages properties [3, 10]. However, it is very important, that the presented models do not consider the cross-cutting mechanisms problem.

IV. PROPOSED METHOD

Based on the concepts, models and principles of ES HLD-methodology [11–14], the following method of analysis of design entities and implementation languages is proposed.

ES design process should be carried out within aspect approach [15–17]. In the initial step, system architect allocates important design space segments (aspects). Each aspect reflects a particular problem space in the project execution. Within a single aspect, the sets of design space axes (subspaces) are allocated. An axis is the certain problem within the project. On each axis, the set of computational (and other) mechanisms, ranged by the certain criteria, are located. The mechanisms provide the means to solve design problems.

Computational mechanism (CM) is the central concept of the proposed method. It is an architectural pattern that demonstrates the principles of computational process organization. In contrast with the popular concept of "design pattern" that does not have fixed requirements for abstractness of description and internals demonstration, CM has to transparently provide with useful "computational" technical principles without fixating of their implementation. Thus, CM should be considered as a specific category of patterns of computer systems design. Along with computational mechanisms, other categories of mechanisms are used, e.g. mechanisms of interaction, verification, debug. Thus, the mechanism is the universal element that can be allocated both within a single design language and across several layers, which involve several languages to work with. The marks on the axes that are proposed within the certain methodologies [6–9], can be treated as the variants of the mechanisms, while the proposed axes can be used as design space axes.

During ES implementation, the set of aspects, design space axes and mechanisms within each axis is used by developers as a library of design decisions, primarily, at the conceptual level [18]. Also, the models that are constructed in these terms can be used at the verification step [19].

Annotation of the source text of the project (primarily multi-language) is proposed to enable the automated support for this approach. The tag language, based on comments of the special format, has been developed. Annotated code allows the fast navigation through mechanisms implementation fragments to be carried out that simplifies manual control over their implementation correctness.

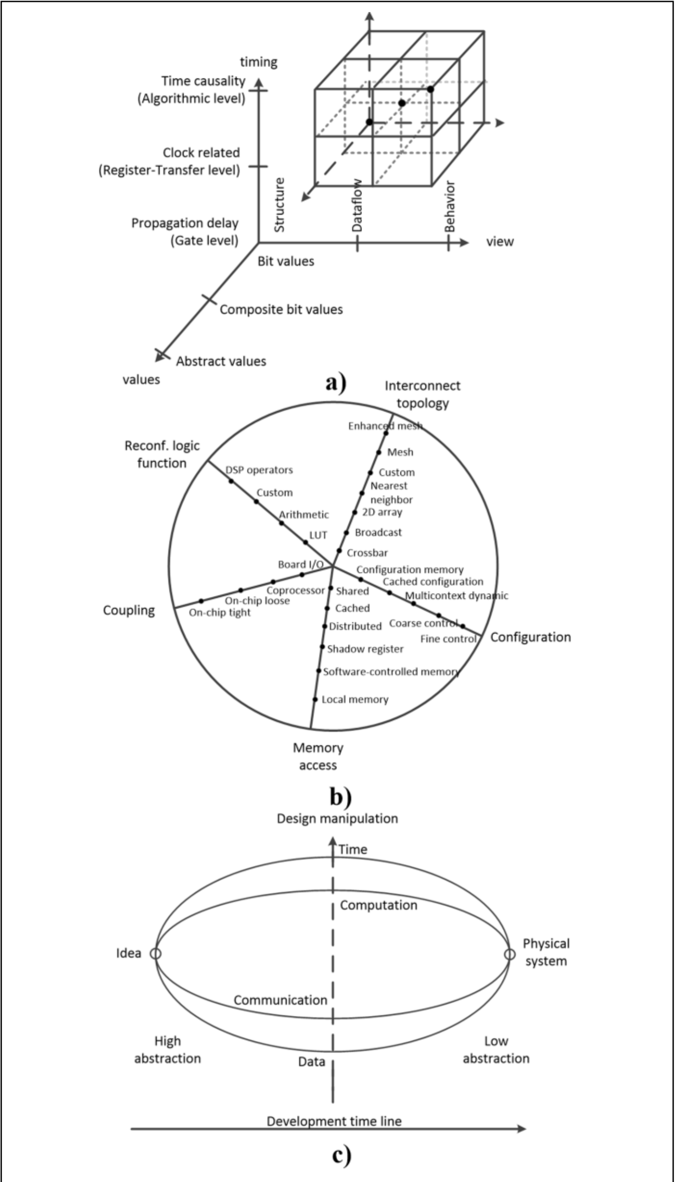

Figure 1. Variants of aspect and design spaces representation in embedded systems design and analysis methodologies.

V. THE USE CASE OF THE METHOD

The proposed method has been applied to the typical set of languages that are used for ES design and development with utilization of programmable processors and dedicated hardware units implemented in FPGA or ASIC (see Fig. 2 and Table 1).

The set of design space axes has been allocated for analysis. This set includes four axes from ESL taxonomy [3, 9] and two extra axes that have been added: data flow / control flow ratio of structural unit functional implementation and axis of functional verification mechanisms. Mechanisms of first extra axis variously combine involvement of instruction and data streams in computational process control. The second axis contains the mechanisms that can be used for computer system testing and verification during its design, depending on the scale of the element under verification. The number of marks on the concurrency and communication axes has been reduced and only important mechanisms, which are not tied to specific implementation, have been left.

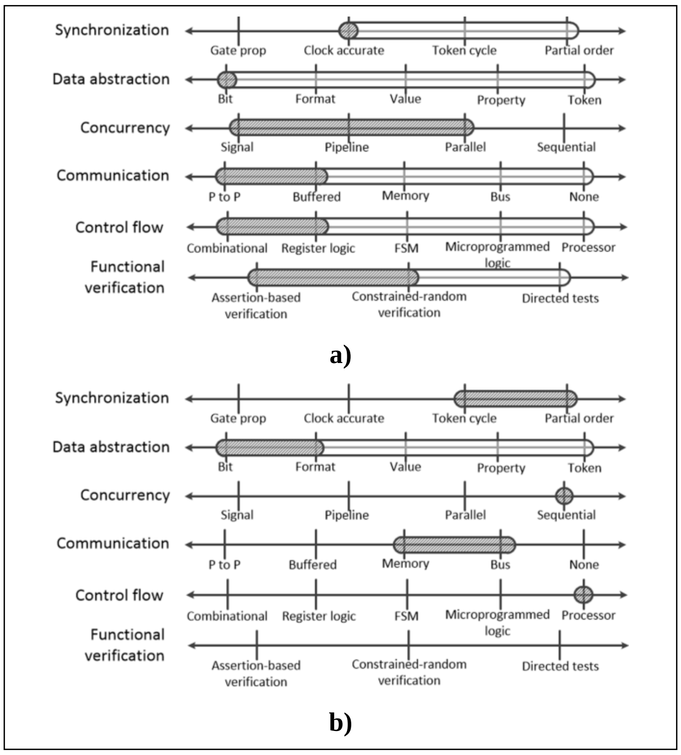

Figure 2. Mechanisms of languages: a) SystemVerilog (synthesizable subset + verification) and b) generic assembler. Shaded outlined areas – built-in mechanisms, outlined areas – mechanisms which can be implemented on the basis of built-in mechanisms.

Generic assembler (assembler of abstract programmable sequential processor core) reflects the capabilities of software implementation and is not tied to specific architectures or processor configurations. If needed, the set of language mechanisms can be extended through processor specialization (by redesign, IP-core configuration or custom extensions). The examples are Microblaze (Xilinx), NIOS (Altera) and other processor cores.

Languages have built-in support of the mechanisms of a certain complexity (or abstractness) and, in most cases, means for combining simple mechanisms to form complex ones. It is assumed that the level of mechanisms' complexity within design space axis can be increased in case mechanism implementation fits into language capabilities. Within this approach, the languages have been analyzed from two perspectives – from the viewpoint of the mechanisms that have built-in support in the language and from the viewpoint of the mechanisms that can be effectively realized by the language means based on built-in mechanisms.

TABLE I. BUILT-IN MECHANISMS OF LANGUAGES.

Design space axes	**Languages**				
	System-Verilog (synthe-sizable)	***System-Verilog (for simu-lation)***	***SystemC (for simu-lation***	***Generic assem-bler***	***SysML (modeling only)***
Syn-chroni-zation	Cycle-accurate	Cycle-accurate, system events, partially ordered	Cycle-accurate, system events, partially ordered	Instr. cycle	Partially ordered (sequence, activity diagrams)
Data ab-strac-tion	Bit and format (with wire and reg vectors)	All (with enums and struc-tures)	All (with structures and classes)	Bit	From format to token (package diagrams)
Con-cur-rency	Signal and block parallelism	Signal and block paralle-lism	Signal, block, software processes (with sc_ process_ handle)	Se-quen-tial	Signal, block, multi-application (with internal block diagram)
Com-muni-cation	P to P and buffered (with wires and regs)	Same as synthesiz able	P to P, buffered, memory (with pointers)	No	No specific mechanism
Control flow	Comb. and register logic	Same as synthesiz able	Same as System-Verilog	Pro-process or as a plat-form	FSM (state-charts), sequential (sequence diagrams)
Func. veri-fication	No	Assertion -based, const-rained-random	No	No	No

The example of using multi-language source code annotation is demonstrated in Fig. 3.

```
File: .\asm\boot.asm, line: 176
CALL P_LOAD_CRC # load from coprocessor to ACC
SUB  CRC_REG    # checking CRC
JEQ  LABEL_BLOCK_CRC_OK
...
File: .\asm\asm.py, line: 198
"RWRK": [11, "R"], # read coprocessor cmd
"WWRK": [12, "R"], # write coprocessor cmd
...
File: .\hdl\wrk.sv, line: 236
if (ctrl_a == ADDR_CRC) begin
    ctrl_do <= crc;
end
...
File: .\hdl\crc32.sv, line: 35
always @* begin
    crc_new[0] = crc_old[0] ^ crc_old[6] ^ ...
```

Figure 3. Fragment of CRC implementation mechanism.

Here, the fragment of cross-cutting implementation of cyclic redundancy checksum (CRC) mechanism is demonstrated. CRC is used for integrity control of bootable software images for heterogeneous multiprocessor system. The system has been implemented as system on chip (SoC). Embedded boot manager is a specialized processor being programmed in assembly language. The cut has been acquired from the annotated code automatically. The source code has been realized in assembly (program for the processor, boot.asm), Python (compiler, asm.py), SystemVerilog (CRC coprocessor, wrk.sv and crc32.sv).

VI. FUTURE WORK

The proposed approach is not formal, thus, quality of its application depends greatly on expert's qualification. Thus, further refinement of the used concepts has to be carried out. Also, extraction of individual subspaces with clearly defined axes and mechanisms sets has to be done. Such axes could be recommended as typical for certain class of projects and problems. The mechanisms that require cross-level implementation are of special interest.

CONCLUSION

The proposed method allows design process to be partially formalized using HLD-methodology system of concepts. Computational mechanism is the central abstraction. Design languages comparison, carried out on the basis of the proposed method and applied to the languages being used in complex hardware-software projects, allows design tools to be chosen more effectively for subtasks and subsystems. The demonstrated approach makes the developers to concentrate on "cross-cutting" conceptual mechanisms and enables control over the adequacy of their multi-stage implementation.

REFERENCES

[1] J. Teich, "Hardware/software codesign: the past, the present, and predicting the future", Proceedings of the IEEE, 2012, vol. 100, pp.1411 – 1430.

[2] D. Densmore, R. Passerone, A. Sangiovanni-Vincentelli, "A Platform-Based Taxonomy for ESL Design", IEEE Design and Test of Computers, September 2006.

[3] B. Bailey, G. Martin, "ESL models and their application", New York: Springer Publication, 2010.

[4] A. Sangiovanni-Vincentelli, "Quo vadis SLD: reasoning about trends and challenges of system-level design", Proceedings of the IEEE, 95(3), 2007, pp.467-506.

[5] M. P. Ward, "Language-Oriented Programming", Software - Concepts and Tools 15(4): 147-161 (1994)

[6] W. Ecker, M. Hofmeister, "The design cube - a model for VHDL designflow representation", Proceedings of the European Design Automation Conference (EuroDAC), Hamburg 1992, 752-757.

[7] A. Chattopadhyay, "Ingredients of adaptability: a survey of reconfigurable processors", VLSI Design, 2013, vol. 2013, p.18.

[8] A. Jantsch, S. Kumar, A. Hemani, "A Metamodel for Studying Concepts in Electronic System Design", IEEE Design & Test of Computers, vol. 17, no. 3, pp. 78-85, Jul. 2000.

[9] B. Bailey, G. Martin, A. Piziali, "ESL Design and Verification: A Prescription for Electronic System Level Methodology", Elsevier Morgan Kaufmann, 2007.

[10] Panagopoulos, G. Papakonstantinou, N. Alexandridis, and T. El-Ghazawi, "A comparative evaluation of models and specification languages for Embedded System design", Languages, Compilers, and Tools for Embedded Systems (LCTES-03), San Diego, Ca., June 11-13, 2003.

[11] A. Platunov, A. Nickolaenkov, and A. Penskoy, "Architectural representation of embedded systems", 2012 Mediterranean Conference on Embedded Computing (MECO), June 2012, pp.80-83.

[12] A. Platunov, A. Kluchev, and A. Penskoi, "HLD Methodology: The Role of Architectural Abstractions in Embedded Systems Design", 14th GeoConference on Informatics, Geoinformatics and Remote Sensing, 2014, pp. 209–218.

[13] Platunov A., Kluchev A., Penskoi A., "Expanding Design Space for Complex Embedded Systems with HLD-methodology", Proc. of the 6th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT) - 2014, pp. 253-260.

[14] A. Platunov, A. Penskoi, and A. Kluchev, "The Architectural Specification of Embedded Systems", 2014 3rd Mediterranean Conference on Embedded Computing (MECO), June 2014, pp. 48-51.

[15] D. Broman, Ed. A. Lee, S. Tripakis, and M. Toerngren, "Viewpoints, formalisms, languages, and tools for cyber-physical systems", 6th International Workshop on Multi-Paradigm Modeling - MPM'12, October 2012, pp.49–54.

[16] A. Platunov, and A. Nickolaenkov, "Aspects in the design of software-intensive systems", 2012 Mediterranean Conference on Embedded Computing (MECO), June 2012, pp.84-87.

[17] J. M. P. Cardoso, P. C. Diniz, J. G. de F. Coutinho, and Z. M. Petrov, "Compilation and Synthesis for Embedded Reconfigurable Systems: An Aspect-Oriented Approach (Google eBook)", Springer, 2013, p. 215.

[18] Kustarev P., Bikovsky S., Antonov A., Yanalov R., "Process control and synchronization patterns for SOC", 14th GeoConference on Informatics, Geoinformatics and Remote Sensing, 2014, Vol. 1, No. 2, pp. 287-294.

[19] Kustarev P., Bikovsky S., Pinkevich V., "Functional monitoring of SoC with dynamic actualization of behavioral model", unpublished.