INFORMATICA, 2004, Vol. 15, No. 1, 111–126

Soft IP Customisation Model Based on Metaprogramming Techniques

Vytautas ŠTUIKYS, Robertas DAMAŠEVIČIUS
Software Engineering Department, Kaunas University of Technology
Studentų 50, LT-3031 Kaunas, Lithuania
e-mail: vystu@if.ktu.lt, damarobe@soften.ktu.lt

Received: June 2003

Abstract. We propose a layered Soft IP Customisation (SIPC) model for specifying and implementing system-level soft IP design processes such as wrapping and customisation. The SIPC model has three layers: (1) *Specification Layer* for specification of a customisation process using UML class diagrams, (2) *Generalisation Layer* for representation of a customisation process using the metaprogramming techniques, and (3) *Generation Layer* for generation of the customised soft IP instances from metaspecifications. UML allows us to specify customisation of soft IPs at a high level of abstraction. Metaprogramming allows us to manage variability in a domain, develop generic domain components, and describe generation of customised component instances. The usage of the SIPC model eases and accelerates reuse, adaptation and integration of the pre-designed soft IPs into new hardware designs.

Key words: soft IP design, reuse, customisation, metaprogramming, UML, generation.

1. Introduction

Ever-increasing complexity of *Systems-on-Chip* (SoC), combined together with the requirements to shorten time-to-market and the need to reduce design costs force hardware (HW) designers to reuse the pre-designed *Intellectual Property* components (IPs). The term denotes an original component that was created by a human designer, is portable, and can be used by other designers. *Soft IPs* are IPs that are described using a high-level HW description language (HDL) such as VHDL, Verilog or SystemC. They are valuable assets for their developers, providers and SoC designers.

Soft IP-based design raises importance of *system-level design processes*, such as IP customisation and wrapping, additionally to *Register Transfer Level* (RTL) design processes, which are concerned with IP design from scratch. A *design process* is a series of commonly used well-defined domain-specific actions and methods performed to achieve a certain well-formulated design aim. The role of system-level design processes is to prepare soft IPs for reuse and integration into a larger HW system or SoC. Currently, HW designers use system-level design processes only casually. However, their counterparts in software (SW) engineering, the *design patterns* (Gamma *et al.,* 1995) described using UML (Booch *et al.,* 1998) diagrams, have attracted much more research and have

increased SW design quality and productivity. We hope that our research will contribute in this area, too.

We implement system-level design processes using parameterisation-based and generative mechanisms provided by languages, which support the metaprogramming (MPG) paradigm. In general, MPG allows raising the level of abstraction in design, thus enables to achieve a great deal of flexibility and reusability. It opens wide capabilities for implementing customisation and adaptation of the pre-designed soft IPs. Furthermore, MPG allows implementing the *multi-dimensional separation of concerns* (Ossher and Tarr, 2000) in soft IP design explicitly, thus improving comprehensibility, facilitating soft IP evolution and reuse, as well as simplifying soft IP integration into new HW designs. In the past, MPG has played a significant role in creating first SW generating systems such as compiler generators. We believe that MPG can also contribute to further increase in HW design productivity, as it did in SW design. Another motivation for the increased interest in the MPG systems is the effort to find an adequate response to the growth of HW design complexity and requirements to increase quality of the designed systems. The knowledge about MPG gained in research of the formal logic programming systems (Sheard, 2001) can contribute to the solution of this problem. This paper continues our previous research (Štuikys *et al.*, 2002b) on the usage of MPG for soft IP design.

In order to tackle with ever-growing complexity of HW design domain, the researchers usually propose high-level models that allow to better understanding the domain, describe design problems in an abstract way and implement design solutions using a verified design methodology. To support system-level design processes in soft IP design, we present a layered *Soft IP Customisation* (SIPC) model and its implementation based on the MPG techniques. We argue that the built-in MPG capabilities of HDLs, such as *templates* (SystemC) or *generics* (VHDL) usually are not enough to implement customisation. We suggest to apply the *heterogeneous* MPG based on the usage of an external metalanguage for parameterisation of HDL programs.

A contribution and novelty of this paper is as follows: (1) the SIPC model, a novel concept for system-level soft IP design, (2) the soft IP customisation framework based on the SIPC model, and (3) analysis of the MPG techniques used for soft IP design.

The paper is organised as follows. In Section 2, we review related research. In Section 3, we explain the principles of MPG, describe the structure of the SIPC model, and present the soft IP customisation framework based on the model. In Section 4, we present the results of our case study. In Section 5, we evaluate our approach. Finally, we conclude in Section 6.

2. Related Works

Soft IP reuse methodologies and models are discussed in numerous papers. For example, Seepold (1999) focuses on the extension of existing methodologies and generalisation. He considers the need for new innovative methodologies to drive IP reuse. Vermeulen *et al*. (2000) propose a system-level IP reuse methodology. Designs are described in three

layers of operations, which allow structural and behavioural reuse. A reusable description consists of the combination of the entities at the three layers. Jacome *et al.* (1999) propose developing the IP libraries using *design space layers* implemented on the top of conventional reuse libraries. Each layer is characterised by structural and behavioural descriptions of IPs, design requirements and constraints. Böttger *et al.* (1998) present an object-oriented model for IP reuse in HW design. The model has three layers. (1) *Specification layer* captures knowledge about a target system in terms of functional and structural requirements. (2) *Function layer* contains knowledge about the functional properties of the domain. (3) *Architectural layer* represents knowledge about possible implementations of system functions.

Several authors see the answer to the IP-based design problems in higher abstraction levels. For example, Chou *et al.* (1999) use higher-level design abstractions to automate the generation of error-prone design details thus allowing the designers to focus on global architectural and functionality decisions. Zhu (2001) proposes a new RTL abstraction, called MetaRTL, which extends traditional HDLs and provides IP reuse at a higher level of design. Meguerdichian *et al.* (2001) present an IP development approach, which considers soft IP design and optimisation at a higher level than standard HDL specifications. Mihal *et al.* (2002) present a methodology that allows mapping high-level application models onto architectures while preserving the application model semantics. The methodology allows for several abstraction layers, thus giving more design flexibility and enabling exploration of a wide array of architectures.

Other papers deal with design of highly parametric soft IPs. Garino *et al.* (1999) emphasise the need for parameterisation of soft IPs to achieve higher flexibility, customisability, reusability, and propose the parametric soft IP design methodology based on VHDL *Generic* statement. Agaësse and Laurent (1999) claim that even the most frequently used soft IPs, although they may implement a well-established standard, require certain customisation to fit an application. Givargis and Vahid (2000) consider parameterisation of HW architectures in terms of power/performance/area requirements. In another paper, Givargis and Vahid (2002) discuss methods that support tuning of the parameterised SoC platforms for optimal SoC performance and power usage. The methods range from simple parameter configuration to aggressive architectural transformations. Fin *et al.* (2001) present a parametric soft IP generation methodology, which allows customising processors with regard to the instruction set and data size by annotating the already designed VHDL descriptions with *command-comments*.

Several authors investigate the problems of soft IP integration into SoC designs. For example, Chang *et al.* (1999) propose a platform-based approach for SoC design. A platform is a family of architectures that satisfy a set of architectural constraints imposed to allow the reuse of HW and SW components. Sangiovanni–Vincentelli (2000) explores methods for selecting families of SW and HW architectures that allow a substantial design reuse. The author also discusses some paradigms for embedded system design that are likely to become the pillars of future tools and flows such as *orthogonalization of concerns*, i.e., separation of various design aspects to allow effective exploration of alternative solutions. Zhang *et al.* (2001) focus on safe integration of soft IPs using XML-based

representation. Automation of this process allows speeding up the system design by using the parameterised component libraries and design frameworks. Koch (2002) proposes an environment for automatic integration of parameterised HW objects, which adhere to a standard interface.

We summarise the related works as follows. The authors emphasise the role of parameterisation, customisation and integration of the pre-designed soft IPs in HW design, and seek for the higher levels of abstraction (design space layers, languages, models, frameworks, platforms), which should ensure higher design quality, productivity and reuse. However, there is a lack of the systematic approach and wider usage of MPG and generative techniques. To address these problems, we present the *Soft IP Customisation* (SIPC) model.

3. Soft IP Customization Model

The purpose of the SIPC model can be stated as follows: *once developed, soft IP could be reused with or without adaptation across the different application domains*. We apply the SIPC model as a framework for developing soft IP generators, which allow us to achieve the following. (1) To *represent* basic domain functionality, (2) To *express* the commonalties and variations of a domain functionality, (3) To *analyse* soft IPs, (4) To *specify* their customisation, and (5) To *generate* the customised target instances (subsystems) destined for further integration into higher-level HW systems.

We explain the SIPC model by describing (1) the usage of UML for specifying customisation, (2) the usage of the MPG techniques for implementing generalisation and generation, (3) describing the structure of the SIPC model layer by layer, and (4) presenting the soft IP customisation framework based on the SIPC model.

Selection of a specification language to specify customisation is a complex problem. Here we must deal with several aspects as follows. (1) A specification language must be a standard and widely known one in order to be understandable by a majority of the designers and to ease communication between different design teams. (2) Specification must be abstract enough in order to be usable for a wide array of architectures and design processes in the domain. (3) Availability of standard tools is a great advantage.

UML (*Unified Modelling Language*) (Booch *et al.*, 1998) is a high-level system specification and modelling language that satisfies these requirements. In the next sub-section, we describe the usage of UML for specifying a customisation design process.

3.1. *Specification of Customisation Process Using UML*

UML combines the usage of the object-oriented design concepts for structural design of a system using *class* diagrams alongside with the behavioural system models (*state*, *sequence* and *activity* diagrams). However, for black-box customisation, we use class diagrams only.

We describe a customisation design process using UML class diagrams as follows. (1) We specify an original soft IP. (2) We specify a target system, a part of which will

be the original soft IP. (3) The customisation process then can be defined as a *transformation* between an original component and a target system, and implemented using the metaprogramming techniques.

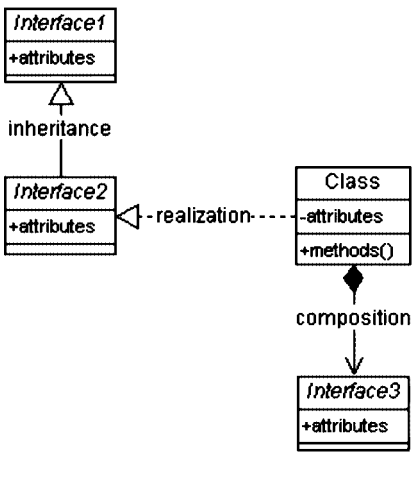
To implement customisation, first, we must define a correspondence between the object-oriented design concepts used in UML class diagrams and the HW design concepts described using a domain language such as VHDL (see Fig. 1).

Elements of UML class diagrams are classifiers, relationships and features. *Classifiers* are interfaces and classes that describe basic design blocks. We specify a VHDL *entity* with an *abstract class* (*interface*) in UML. VHDL *architecture* is specified using a *class.*

Relationships describe different types of connections and associations between classifiers. The *inheritance* relationship means that a VHDL entity inherits the I/O ports from a base entity (there is no corresponding abstraction in VHDL). The *composition* relationship describes composition of a system from the components and corresponds to a VHDL *port map* statement. The *realisation* relationship corresponds to a VHDL *entity-architecture* pair.

Features describe parameters, attributes and methods of classifiers. VHDL ports are specified using *public attributes* (marked by '+') of a class, and VHDL signals – using *private attributes* (marked by '−'). VHDL *processes* and *procedures* are specified using class *methods.*

More details about specification of HW design processes using UML can be found in (Damaševičius and Štuikys, 2004). Once the customisation design process has been specified, a designer must describe how it can be implemented automatically. Our approach deals with a general problem – the need of a *description language for representing architectures of the generic soft IP families.* Capabilities of the standard HDLs for flexible parameterisation, expressing variations in a design, as well as customising components




```
-- VHDL entity corresponds to UML interface
ENTITY  Interface1 IS
  PORT (
-- public attributes are used as VHDL ports
  );
END  Interface1;

ENTITY  Interface2 IS
  PORT (
-- Interface2 inherits some ports from Interface1
  );
END  Interface2;

-- VHDL architecture corresponds to UML class
ARCHITECTURE  Class  OF  Interface2  IS
-- Class provides an implementation for Interface2

-- private attributes are used as VHDL signals
  BEGIN
-- processes are described using UML class methods
        method: PROCESS()
                  BEGIN ... END PROCESS;

-- Class contains component Interface3
        p1: Interface3 PORT MAP ( ... );
END  Class;
```

(b)

Fig. 1. Correspondence between UML and VHDL: a) UML class diagram, and b) VHDL abstractions.

to the context of their usage are usually not enough. We assume that these problems can be solved in terms of MPG paradigm, which we consider in the following subsection.

3.2. *Introduction to MPG Techniques*

In general, MPG is a higher-level programming technique, which provides a means for manipulating with domain programs as data. The main aim of MPG is to create a *metaspecification* (aka metaprogram) – a specification of a program generator for a narrow application domain.

A metaspecification consists of a generic interface and a family of related domain program instances encapsulated with their modification algorithm. A generic interface describes the generic parameters of a metaspecification. The modification algorithm describes generation of a particular instance depending upon values of the generic parameters. The modification algorithm ranges from simple metaconstructs such as *meta-if* (conditional generation) and *meta-for* (repetitive generation) to the sophisticated application-specific metapatterns, which are composed of the nested combinations of the simpler metaconstructs. As a metaspecification is a concise representation of its instances, it can be treated as a generic component, too.

We develop a metaspecification in several steps: (1) Domain (usually represented by one or more available domain component instances and requirements for their modification) is *analysed*. (2) Modification concerns are *identified* and *separated*. These concerns represent the *variable* aspects in a domain, which *depend* upon generic parameters and require the MPG techniques to be applied. (3) Modification concerns are expressed through generic parameters and *implemented* using the MPG techniques. (4) Modification concerns are *integrated* with the *fixed* domain aspects (i.e., domain functionality described using a domain language), which are *orthogonal* with respect to the values of the generic parameters. Such metaspecification describes commonalties and variations in a particular application domain.

The MPG abstractions can be categorised into two large groups as follows: the *homogeneous* and *heterogeneous* ones. The form and way how the concerns are separated underscores the essential differences between these MPG paradigms. At the core of the homogeneous MPG is *implicit* separation of concerns. At the core of the heterogeneous MPG is *explicit* separation of concerns.

Heterogeneous MPG (Fig. 2) is based on the usage of two different languages in the same metaspecification. The lower-level language (*domain language*) expresses basic domain functionality. The higher-level language (*metalanguage*) expresses generalisation and describes domain program modifications. Using a metalanguage as a higher-level abstraction, the different domain language component instances are woven together and make up a metaspecification. The metaspecification is then used as a set of instructions for its environment (processor, compiler) to generate the specific component instances in a domain language.

In *homogeneous* MPG (Fig. 3), we have two "different" subsets of the same language, the lower-level (*domain-oriented*) and higher-level (*template, generic* or *meta*) ones, and

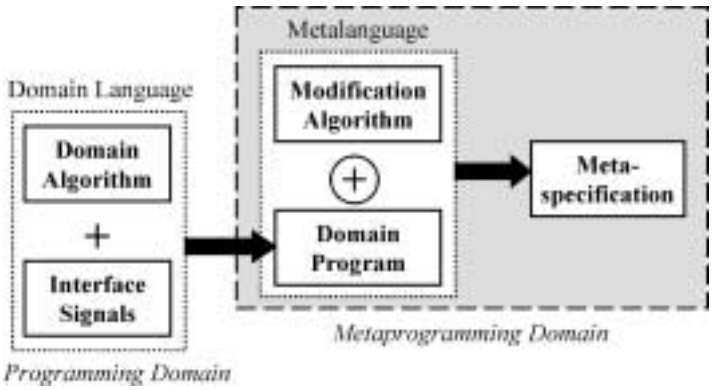


Fig. 2. Framework of heterogeneous MPG.

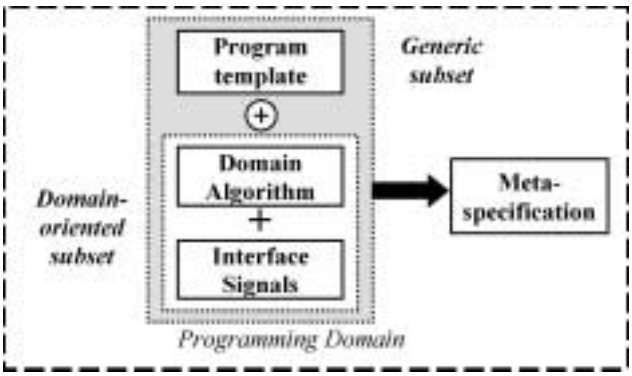


Fig. 3. Framework of homogeneous MPG.

just one design environment. The domain-oriented subset expresses basic domain functionality. The generic subset expresses generality in a domain. The decomposition of a given language into the subsets and specifying the content for each subset are subtle problems. Therefore, one can argue that there is a thin line between programming and homogeneous MPG. Especially, it is true with VHDL, because the language designers have not explicitly defined which constructs are the higher-level ones. In our view, the generic constructs in VHDL are *packages, constants, generics*, and *if/for generate* statements.

Application of the homogeneous MPG techniques is hindered by the limited expressiveness of the domain language syntax. For example, the existing parameterisation mechanisms in VHDL allow only changing the width of the I/O signals, but can not specify more complex modifications such as the inclusion/exclusion of the signals, the usage of the different communication schemes, or inheritance. On the contrary, the heterogeneous MPG techniques are domain language-independent and have virtually unlimited capabilities for modifying domain language code. Therefore, we argue that heterogeneous MPG is more suitable for customisation of soft IPs.

3.3. *Structure of SIPC Model*

We describe the structure of the SIPC model at three layers: (1) *Specification Layer*, (2) *Generalisation Layer*, and (3) *Generation Layer*. We illustrate the roles of the layers by presenting a solution to a simple soft IP customisation problem: the automatic extension of soft IPs with control signals (*clock*, *enable*, and *reset*). This problem is trivial, however, it has a wide application in HW design (see, e.g., *clock gating* (Li *et al*., 2003), *input latching* and *input gating* (Williams *et al*., 2000) techniques for low power design, *pipelining* (Marinescu and Rinard, 2001) for efficiency) and allows us to demonstrate our approach systematically.

3.3.1. *Specification Layer*

Specification Layer (Fig. 4) is dedicated to specifying the customisation design process, i.e., adaptation of soft IP to the needs of a particular designer and integration into the designed HW system. The role of Specification Layer is (1) to *analyse* (*parse*) a soft IP, (2) to *specify* an application-specific *Customisation model* in a semi-formal way, and (3) to *implement* the automatic extraction of the application-specific parameter values for further customisation of soft IP.

Parsing is an application-specific domain analysis method that is concerned with automatic analysis of abstract domain representations – source code of domain language programs. Parsing decomposes a domain language specification according to the domain language syntax into an *Abstract Syntax Tree* (AST). AST then is used for extraction of the application-specific domain information for further customisation. The aim is to streamline selection of the context-dependent parameter values for metaspecifications.

We specify the Customisation model for given application using UML *class diagrams* as described in sub-section 3.1. In Fig. 5, we present the specification for our soft IP customisation problem and explain it below as follows. An *interface* SoftIP describes the VHDL *entity* of the original soft IP, which can be extended by the additional I/O ports using *inheritance*. The *realisation* relationship describes the implementation (VHDL *architecture*) of the corresponding interface (entity). The functionality is specified using class methods (VHDL *processes* or *procedure*s). The *aggregation* relationship shows that

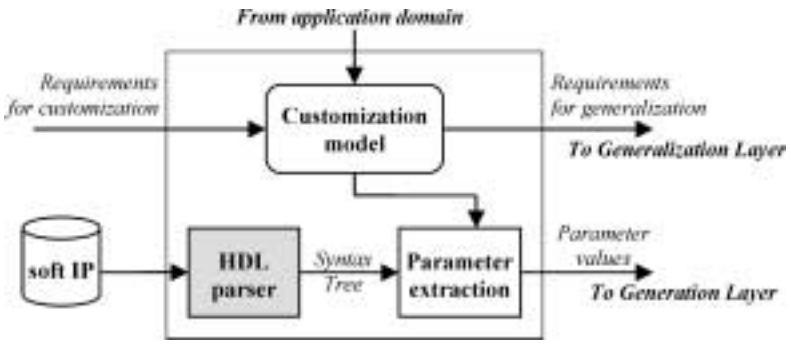


Fig. 4. Structure of Specification Layer.

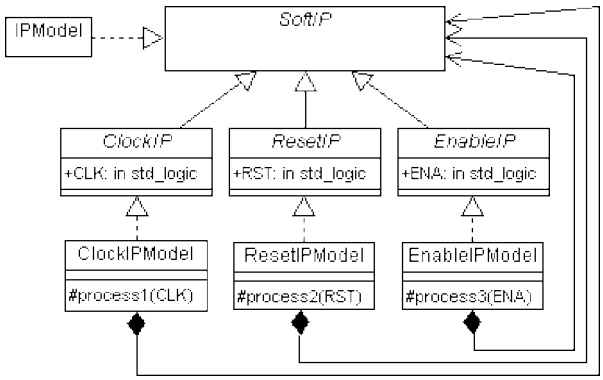


Fig. 5. Customisation model for extending soft IPs with control signals.

the additional functionality, which implements, e.g., clocking, is composed with the original soft IP using VHDL *port map* statement. The presented customisation model is an example of application of a *Wrapper* design pattern, which we examined in (Damaševičius *et al.,* 2003).

3.3.2. *Generalisation Layer*

Generalisation Layer (Fig. 6) is dedicated for representing the *generic components.* A generic component is a metaspecification, which uses the parameterisation-based mechanisms to encapsulate a family of the related domain component instances (e.g., generic gate, generic ALU, etc.). By the family, we mean a collection of the different instances, which have similar functionality, but different (application- or technology-oriented) characteristics.

The domain component instance is a structure that consists of *interface*, which is used for communicating with other instances, and *functionality*, which implements the functional requirements and is described using standard HDL abstractions. Note that in

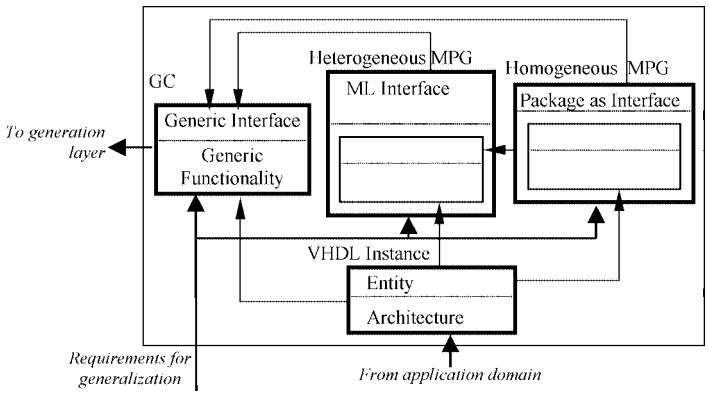


Fig. 6. Structure of Generalisation Layer.

```
$
"Specify VHDL entity ports for IP:"   { }     ports  := none;
"Add a clock signal to the entity?"   {0, 1}  add_cl := 1;
"Add a reset signal to the entity?"   {0, 1}  add_rs := 1;
"Add an enable signal to the entity?" {0, 1}  add_en := 1;
$
ENTITY customized_IP IS
  PORT (
@if[add_cl = 1, {
        CLK: in STD_LOGIC;}]
@if[add_rs = 1, {
        RST: in STD_LOGIC;}]
@if[add_en = 1, {
        ENA: in STD_LOGIC;}]
@if[ports neq {none}, {
        @sub[ports]
}] );
END customized_IP;
```

Fig. 7. Example of PROMOL metaspecification for extending soft IPs with control signals.

VHDL, these parts of the component are called *entity* and *architecture*, respectively.

Generic components can be described using generic abstractions of a standard HDL (*homogeneous* MPG) or an external metalanguage (*heterogeneous* MPG). We can use a dedicated (e.g., pre-processing, macro) language or a common programming language, such as Java or C++, as a metalanguage. The generic interface serves for representing the generic parameters of a generic component. To demonstrate heterogeneous MPG, we use Open PROMOL (Štuikys *et al.*, 2002a) as an external metalanguage. In Fig. 7, we present a PROMOL metaspecification, which demonstrates the usage of *meta-if* (***@if*** function) to conditionally include the I/O ports into the VHDL entity. Note that the '**$**' symbols denote the generic interface of a PROMOL metaspecification. The value of the generic parameter *ports* is usually not specified manually by a designer, but rather extracted automatically from the soft IP description.

3.3.3. *Generation Layer*

Generation Layer (Fig. 8) is dedicated to processing the generic components (metaspecifications) and generating the customised instances of domain components. A designer or an external tool specifies the particular instances through selection of the generic pa-

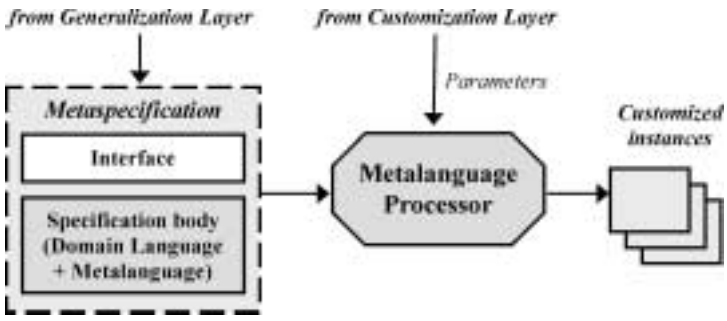


Fig. 8. Structure of Generation Layer.

```
ARCHITECTURE  model  OF  alu_clk  IS
  SIGNAL a_clk: std_logic_vector (31 downto 0);
  SIGNAL b_clk: std_logic_vector (31 downto 0);
  SIGNAL op_clk: std_logic_vector (3 downto 0);
  SIGNAL ci_clk, mode_clk: std_logic;
BEGIN
    p1: alu PORT MAP (a_clk, b_clk, op_clk,
              ci_clk, mode_clk, cout, ovf, z);
    process1: PROCESS (clk)
      BEGIN
        if (clk'event and CLK='1') THEN
            a_clk <= a;
            b_clk <= b;
            op_clk <= op;
            ci_clk <= ci;
            mode_clk <= mode;
        END IF;
    END PROCESS;
END model;
```

Fig. 9. Example of generated VHDL code.

rameter values. The generation process is performed automatically using a metalanguage processor. It is important to emphasise that the conventional compilers of general-purpose programming languages, such as C++ and Java, can be used for implementing the generation process, because these languages can be used as metalanguages, too.

Generation Layer does not introduce any new domain functionality, but rather provides a means for handling the variations encapsulated in the generic components and performing the customisation of soft IPs. Generation Layer uses the MPG abstractions, which allow implementing the *parameterisation mechanisms*, and provide the capabilities for adaptation and customisation of the soft IPs to the specific requirements of a designer.

In Fig. 9, we present the generated VHDL architecture, which adds the clocking logic to the third-party soft IP. The functionality of the clock logic is described using a VHDL process *process1* and the soft IP is inserted using a VHDL *port map* statement.

In the next subsection, we present the soft IP design framework based on the SIPC model.

3.4. *Soft IP Customisation Framework Based on SIPC Model*

Suppose that a system designer retrieves a third-party soft IP, analyses it, and discovers that this soft IP needs some customisation in order to integrate it successfully into a particular HW system. Then the designer needs to perform the following actions:

(1) *Analyse* the domain problem, *formulate* the requirements for customisation, and *specify* the customisation model using UML class diagrams.
(2) *Analyse* the available domain component instances, *formulate* the requirements for generalisation, and *develop* the metaspecifications using the selected metalanguage.
(3) *Analyse* the soft IP using a HDL parser, and *extract* the generic parameter values for the metaspecifications.

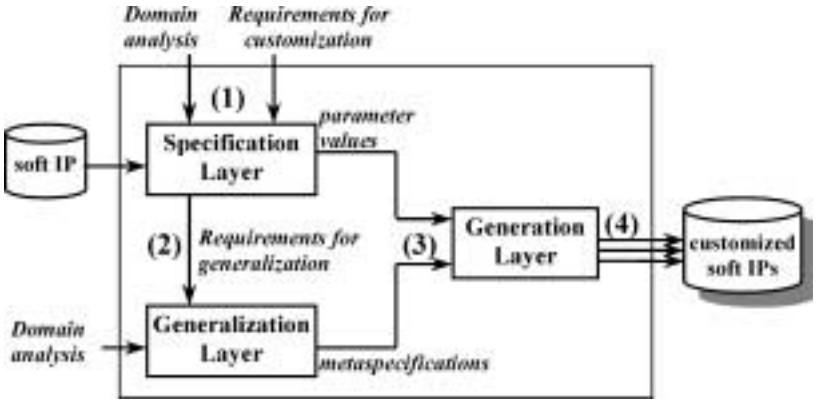


Fig. 10. Design flow of the proposed methodology.

(4) *Generate* the customised soft IP instances from the metaspecifications using a metalanguage processor.

In the SIPC model, the designer performs the design steps (1) & (2) manually. The design steps (3) & (4) are performed automatically. Design flow of the proposed methodology is summarised in Fig. 10.

In the next section, we present the results of our case study.

4. Results of Case Study

In our case study, we have used a variety of public-domain soft IPs: 8b ALU from i8051 micro-controller (Givargis, 2000), 32b ALU from DLX processor (Gumm, 1995), 32b Chang ALU (Chang, 1999), and 32b Booth multiplier (Booth, 2001). Synthesis results (Synopsys tools; CMOS 0.35 um) for the original soft IPs and generated components are presented in Table 1.

Synthesis results show an average increase in chip area of about 38% for added *clock* logic, 11% for added *enable* logic, and 4% for added *reset* logic.

Table 1

Synthesis results

Soft IP	Area, Cells (IP)	Area, Cells (IP+clock)	Increase, %	Area, cells (IP+enable)	Increase, %	Area, cells (IP+reset)	Increase, %
8b i8051 ALU	2265	2789	23.1	2415	6.6	–	–
32b DLX ALU	2280	3164	38.8	2625	15.1	–	–
32b Chang ALU	1945	3044	56.5	2295	18.0	2085	7.2
32b Booth multiplier	7859	10370	32.0	8179	4.1	7942	1.1

5. Evaluation and Discussion

The *problems* of implementing the SIPC model are as follows: (1) analysis of domain applications and extraction of the customisation model for describing customisation of soft IPs, (2) selection of a metalanguage and generalisation of domain objects, and (3) qualification of the customised soft IPs. The key for the SIPC model is generic representation introduced by MPG. It simplifies the control of variations, extends reusability and adds an extra value to the soft IPs.

Homogeneous MPG capabilities of the standard HDLs allow implementing the parameterisation-based structures of the SIPC model only partially. However, these capabilities are not enough because of (1) the lack of flexibility for describing a wide range of modifications required for adaptation to the variety of functional requirements, (2) the limitations of the synthesis tools, (3) the over-generalisation problem, (4) the lack of the explicit separation of concerns when using a single language.

We have suggested to use the *heterogeneous* MPG techniques introduced by the abstractions of an external metalanguage. The main advantages are as follows: (1) independence from a HDL, (2) flexibility in implementing generalisation and modification, (3) adaptability to the synthesis tools and target technology, (4) no over-generalisation (from the user's viewpoint), (5) the explicit separation of concerns.

The strengths of our approach are as follows. (1) The usage of UML allows us to *specify* the customisation problem at a high level of abstraction. (2) The usage of MPG allows us to *manage variability* in a domain, as well as to *generate* the customised soft IP instances *automatically*.

The limitations of our approach are as follows. (1) The UML model does not reflect the *physical constraints* imposed on soft IPs. (2) The metaspecifications so far are *developed manually*, thus requiring considerable programming efforts. (3) The *validation* of the metaspecifications is much more difficult than the validation of the soft IPs.

These limitations could be overcome in a variety of ways. For example, physical constraints can be introduced by adopting extensions of UML for real-time and embedded systems. Development of metaspecifications could be simplified by generating (at least) parts of them automatically. Further research is needed to explore these possible solutions.

6. Conclusions and Future Work

Increasing complexity of HW design requires adoption of the new system-level design methods, which deal with automated customisation and integration of the pre-designed soft IPs. We have proposed a novel *Soft IP Customisation (SIPC) Model*, which is based on the usage of UML and the MPG techniques, and allows implementing application-specific customisations of the soft IPs. The SIPC model allows achieving higher reusability and adaptability of the pre-designed soft IPs.

Future work will focus on overcoming the limitations of our approach, as well as on research and implementation of the customisation models for other application domains such as embedded systems.

Acknowledgements

We thank the anonymous reviewers for their comments that allow us to improve the paper.

References

Agaësse, J.F., and B. Laurent (1999). Virtual components application and customization. In *Proc. of Design, Automation and Test in Europe (DATE'1999)*. pp. 726–727.

Booch, G., I. Jacobson, J. Rumbaugh and J. Rumbaugh (1998). *The Unified Modeling Language User Guide.* Addison–Wesley.

Booth (2001). `http://www.cs.umbc.edu/help/VHDL/samples/`

Böttger, J., K. Agsteiner, D. Monjau and S. Schulze (1998). An object-oriented model for specification, prototyping, implementation and reuse. In *Proc. of Design, Automation and Test in Europe (DATE 1998)*. pp. 303–309.

Chang, H., L. Cooke, M. Hunt, G. Martin, A. Mc-Nelly and L. Todd (1999). *Surviving the SoC Revolution: A Guide to Platform-Based Design*. Kluwer Academic Publishers, Norwell.

Chang, K.C. (1997). *Digital Design and Modeling with VHDL and Synthesis*. IEEE Computer Society Press, The Institute of Electrical and Electronic Engineers, Inc., Los Alamitos, CA.

Chou, P., R. Ortega, K. Hines, K. Partridge and G. Borriello (1999). IPChinook: an integrated ip-based design framework for distributed embedded systems. In *Proc. of 36th Design Automation Conference (DAC'1999)*. pp. 44–49.

Damaševičius, R., G. Majauskas and V. Štuikys (2003). Application of design patterns for hardware design. In *Proc. of 40th Design Automation Conference (DAC 2003)*. pp. 48–53.

Damaševičius, R., and V. Štuikys (2004). Application of UML for hardware design based on design process model. In *Proc. of Asia South Pacific Design Automation Conference (ASP-DAC 2004)*. pp. 244–249

Fin, A., F. Fummi and G. Perbellini (2001). Soft-cores generation by instruction set analysis. In *Proc. of 14th Int. Symposium on System Synthesis (ISSS'01)*. pp. 227–232.

Gamma, E., R. Helm, R. Johnson and J. Vlissides (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison–Wesley.

Garino, P., G. Cesana, M. Paolini, M. Turola and S. Vercelli (1999). Experiences and issues in developing re-usable IP soft cores for the new millenium ICT products. In *IP'1999 Europe*.

Givargis, T. (2000). Intel 8051 Microcontroller.
`http://www.cs.ucr.edu/~dalton/i8051/i8051syn/`

Givargis, T., and F. Vahid (2000). Parameterized system design. In *Proc. of 8th Int. Workshop on Hardware/Software Codesign (CODES'2000)* . pp. 98–102.

Givargis, T., and F. Vahid (2002). Platune: a tuning framework for system-on-a-chip platforms. *IEEE Transactions on Computer Aided Design*, **21**(11).

Gumm, M. (1995). DLX Processor.
`ftp://ftp.informatik.uni-stuttgart.de/pub/vhdl/vlsi_course/vhdl_src/`

Jacome, M.J., H.P. Peixoto, A. Royo and J.C. Lopez (1999). The design space layer: supporting early design space exploration for core-based designs. In *Proc. of Design, Automation and Test in Europe (DATE'1999)*. pp. 676–683.

Koch, A. (2002). Compilation for adaptive computing systems using complex parameterized hardware objects. *Journal of Supercomputing*, **21**, 179–190.

Li, H., S. Bhunia, Y. Chen, T.N. Vijaykumar and K. Roy (2003). Deterministic clock gating for microprocessor power reduction. In *Proc. of the 9th Int. Symposium on High-Performance Computer Architecture (HPCA'03)*. pp. 113–122.

Marinescu, M.-C., and M. Rinard (2001). High-level synthesis of pipelined circuits from modular queue-based specifications. *Transactions of the Institute of Electronics, Information, and Communication Engineers (IEICE)*, **E84-A**(11), 2655–2664.

Meguerdichian, S., F. Koushanfar, A. Mogre, D. Petranovic and M. Potkonjak (2001). MetaCores: design and optimization techniques. In *Proc. of 38th Design Automation Conference (DAC'2001)* . pp. 585–590.

Mihal, A., C. Kulkarni, C. Sauer, K. Vissers, M. Moskewicz, M. Tsai, N. Shah, S. Weber, Y. Jin, K. Keutzer and S. Malik (2002). A disciplined approach to the development of architectural platforms. *IEEE Design and Test of Computers*, **19**, 2–12.

Ossher, H., and P. Tarr (2000). Multi-dimensional separation of concerns and the hyperspace approach. In M. Aksit (Ed.), *Software Architectures and Component Technology*. Kluwer Academic Publishers, Dordrecht.

Sangiovanni–Vincentelli, A. (2000). Platform-based design: a path to efficient design re-use. *Proc. of the First Int. Symposium on Quality of Electronic Design (ISQED 2000)*. pp. 209–210.

Seepold, R. (1999). Reuse of IP and virtual components. In *Proc. of Design, Automation and Test in Europe (DATE'1999)*

Sheard, T. (2001). Accomplishments and research challenges in meta-programming. In *2nd Int. Workshop on Semantics, Application, and Implementation of Program Generation (SAIG'2001), LNCS*, Vol. 2196. Springer. pp. 2–44.

Štuikys, V., R. Damaševičius and G. Ziberkas (2002a). Open PROMOL: an experimental language for domain program modification. In A. Mignotte, E. Villar and L. Horobin (Eds.), *System on Chip Design Languages*. Kluwer Academic Publishers, Boston. pp. 235–246.

Štuikys, V., R. Damaševičius, G. Ziberkas, and G. Majauskas (2002b). Soft IP design framework using metaprogramming techniques. In B. Kleinjohann, K.H. (Kane) Kim, L. Kleinjohann and A. Rettberg (Eds.), *Design and Analysis of Distributed Embedded Systems*. Kluwer Academic Publishers, Boston. pp. 257–266.

Vermeulen, F., F. Catthoor, D. Verkest, and H. De Man (2000). Formalized three-layer system-level reuse model and methodology for embedded data-dominated applications. In *Proc. of Design, Automation and Test in Europe (DATE'2000)*. pp. 92–98.

Williams, A.C., A.D. Brown, and M. Zwolinski (2000). Simultaneous optimisation of dynamic power, area and delay in behavioural synthesis. *IEE Proceedings – Computers and Digital Techniques*, **147**(6), 383–390.

Zhang, T., L. Benini and G. De Micheli (2001). Component selection and matching for IP-based design. In *Proc. of Design, Automation and Test in Europe (DATE'2001)*. pp. 40–46.

Zhu, J. (2001). MetaRTL: raising the abstraction level of RTL design. In *Proc. of Design, Automation and Test in Europe (DATE'2001)* . pp. 71–76.

V. Štuikys is currently a professor at Software Engineering Department of Kaunas University of Technology, Kaunas, Lithuania. He received the PhD and doctor habilitatis titles from Kaunas University of Technology in 1970 and 2002, respectively. He is a teacher and researcher as well as a leader of the research group "Design Process Automation". His research interests include software and hardware design methodologies, IP reuse, component-based programming, metaprogramming and program generation, CAD systems and soft IP design. He has published more than 100 papers (more than 20 in recent years) in the area. He is an author of several books and a monograph.

R. Damaševičius received the BSc degree in 1999 and the MSc degree (cum laude) in 2001 both in informatics from Kaunas University of Technology, Kaunas, Lithuania. Currently he is an assistant at Software Engineering Department and is pursuing the PhD degree at Kaunas University of Technology. His research interests include metaprogramming, software reuse, software generation and program transformation, as well as hardware design with VHDL and SystemC. He has published more than 20 papers in the area.

Lanksčių intelektualiosios nuosavybės komponentų priderinimo modelis, grįstas metaprogramavimo metodais

Vytautas ŠTUIKYS, Robertas DAMAŠEVIČIUS

Šiame straipsnyje mes siūlome lanksčių intelektualiosios nuosavybės komponentų priderinimo modelį, kuris yra skirtas sisteminio lygmens komponentų projektavimo procesų specifikavimui ir realizavimui. Modelis yra sudarytas iš trijų lygmenų. Specifikavimo lygmuo yra skirtas priderinimo proceso specifikavimui, naudojant UML klasių diagramas. Apibendrinimo lygmuo yra skirtas priderinimo proceso atvaizdavimui, naudojant metaprogramavimo metodus. Kūrimo lygmuo yra skirtas priderintų komponentų egzempliorių automatiniam sukūrimui. UML leidžia specifikuoti komponentų priderinimą aukštame abstrakcijos lygmenyje. Metaprogramavimas leidžia valdyti variantiškumą srityje, kurti bendrinius srities komponentus ir aprašyti priderintų komponentų egzempliorių automatinio sukūrimo procesą. Siūlomo komponentų priderinimo modelio taikymas leidžia palengvinti ir pagreitinti anksčiau sukurtų intelektualiosios nuosavybės komponentų atkartojimą, adaptavimą ir integravimą į naujas aparatūrines sistemas.