INFORMATICA, 2003, Vol. 14, No. 1, 3–18

Wrapping of Soft IPs for Interface-based Design Using Heterogeneous Metaprogramming

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

Received: October 2002

Abstract. We discuss the application of heterogeneous metaprogramming (MPG) for the interface-based design, which deals with the synthesis of the communication interfaces between Intellectual Property components (IPs). Heterogeneous MPG is based on the simultaneous usage of a domain language for describing domain functionality and a metalanguage for developing generic components and program generators. We present a design framework based on the MPG techniques. The novelty of our approach is that we apply the concept of heterogeneous MPG for the automatic generation of IP wrappers for communication between the third-party soft IPs systematically. We present a case study for the synthesis of the communication interfaces using a FIFO protocol.

Key words: heterogeneous metaprogramming, interface-based design, interface synthesis, IP wrapper.

1. Introduction

As complexity of *Systems-on-Chip* (SoC) continuously increases, the designers are focusing on reusing and integrating the existing *Intellectual Property* components (IPs) into SoC rather than developing new components from scratch. The vast majority of IPs exists now in the form of the so-called *soft* IP (Keating and Bricaud, 1999). The soft IP is a description of the hardware (HW) component in a high-level HW description language (HDL) such as VHDL, Verilog, or SystemC. The problem of integrating soft IPs that were developed by different IP vendors using different design methodologies and implementation technologies, and use different communication protocols (e.g., handshake, FIFO, etc.) and operating frequencies, is an imperative in SoC design. When such soft IPs have to be integrated into new systems, a very important issue is the adaptation of the IP's interface to a system-specific HW communication model without affecting the functionality of the IP. The soft IP integration issues are addressed by the so-called interface-based design.

The *interface-based design* (Rowson and Sangiovanni–Vincentelli, 1997) is a design methodology that focuses on the *explicit separation* of the soft IP's functionality from the communication aspects of the soft IPs to be composed into a system. The design of a system is carried out in two steps: (a) the design (or selection) of the soft IPs, and (b) the

design of the interaction between these soft IPs through the appropriate communication protocol. This separation allows (1) to explore the soft IP design space by choosing between different communication models, (2) to model the system behaviour independently of the communication, and (3) to reuse the existing communication models.

The main problem of the interface-based design is how to generate (aka *synthesise*) a communication interface in the form of a *glue logic* (aka *IP wrapper*) that allows soft IPs to communicate one with another differently depending on the context of an application. Soft IPs, for example, may communicate asynchronously or synchronously, and each soft IP may operate at the same or different clock frequencies. Furthermore, multiple soft IPs may communicate with a single component through a single I/O port or through many ports. Each of these options presents a unique challenge for automating interface generation. The process of generating interfaces is also known as *wrapping* of the IPs (Lennard *et al.*, 2000). We consider the *wrapping* of the soft IPs for the interface-based design as a *sub-problem* of SoC design here.

The common way to implement the generation of the glue logic is to apply some generative techniques such as *metaprogramming* (MPG) (Sheard, 2001). The standard HDLs have limited support for MPG. Some HDLs (such as VHDL, SystemC) support *homogeneous* MPG only, and thus have limited capabilities for the automatic customisation of the IP interface through the existing parameterisation mechanisms. Other HDLs (such as Verilog) have no MPG capabilities at all. Therefore, the most straightforward solution is a manual modification of the soft IP, which is error-prone and requires detailed knowledge about the implementation of the IP.

The goal of this paper is to present an approach based on the *heterogeneous* MPG techniques in order to automate the process of generating communication interfaces between soft IPs. We formulate the problem under consideration as follows. For a given soft IP, which is considered as a black-box entity, we need to automatically generate a IP wrapper – the communication interface with other soft IP – satisfying a variety of the pre-specified requirements.

A novelty and contribution of our approach is that we apply the heterogeneous MPG techniques for creating IP wrappers systematically. The techniques allow achieving higher flexibility, reusability and adaptability in soft IP design. We present a design framework based on the heterogeneous MPG techniques.

The structure of the paper is as follows. We consider the related works in Section 2. We analyse the soft IP interfaces, describe a concept of the IP wrapper and its generation using the heterogeneous MPG techniques, and present a design framework in Section 3. We present a case study in Section 4. We evaluate the results and present discussion in Section 5. Finally, we conclude in Section 6.

2. Related Works

The researchers propose several related design methodologies such as *interface-based design* (Rowson and Sangiovanni–Vincentelli, 1997), *interface synthesis* (Rajawat *et al.*,

2000), or *communication-based design* (Sgroi *et al.,* 2001). These methodologies, despite their differences, all deal with the problem of inter-connecting IPs, and emphasise the need for the *orthogonalisation of concerns*, i.e., separating the computation (the functional behaviour of IPs) from the communication (interaction between IPs) aspects. The research of the interface-based design can be categorised into four areas as follows.

2.1. *Optimisation*

The interface optimisation deals with the improvement of system's performance by optimising the IP communication at a high level in terms of better utilisation of resources (busses, buffers, control logic, etc.). The optimisation usually involves three problems: (a) sharing of busses and bus control signals between different HW components; (b) scheduling interface operations; and (c) determining the constraints imposed by the communication.

For example, Gutberlet and Rosenstiel (1994) use a subset of VHDL (*procedures*) to describe the interface behaviour by the low-level signal assignments and *wait* statements. By specifying the interface as a sequence of the atomic operations, performance optimisations can be made at the behavioural level, rather than the logic level, which has a greater impact on the system's performance. In the next step, the interface components, which handle the communication between IPs, are generated. Lysecki and Vahid (2002) use a Petri net model to describe a pre-fetching scheme, which allows reducing the performance penalty introduced by the interface synthesis.

2.2. *Interface Specification Languages*

This area of research deals with the development of the interface specification (modelling) languages, or formal methods for generating communication protocols from the high-level specifications. These languages introduce the constructs allowing a rapid prototyping of the synthesizable interface descriptions.

For example, Madsen and Hald (1995) use a formal notation, which describes the communication events, to transform the server-side interface description into the client-side one. Rowson and Sangiovanni–Vincentelli (1997) describe communication at a high level of abstraction using tokens. As the design moves to the implementation phase, a framework is provided for transforming the token passing scheme into actual bus protocols, such as PCI or EISA. Öberg *et al.* (1996) developed *ProGram* language, which provides constructs that are particular to interfaces: port definitions, which declare the directions of interface signals, the width of communication busses and the frequency at which they operate. Communication is described by the patterns of bits that are written to or read from the IP's ports. Grammar rules describe legal operation alternatives based on the state of the interface. Finally, a synthesizable description of the interface is generated. Ortega and Borriello (1998) examine the problem of mapping a high-level specification to an architecture that uses particular bus protocols for communication. Siegmund and Mueller (2000) use the extension of VHDL for the specification and synthesis of the adaptive interfaces for soft IPs. The specifications of IP's functional behaviour and IP's interface are

separated into different design units. The interface unit then can be efficiently adapted to a system-specific communication scheme without the need for a protocol conversion module or the modification of the IP. The Synopsys' *Protocol Compiler*, which is based on the research of Seawright *et al.* (1996), is the most commonly used industrial tool capable to synthesise the interfaces. The Protocol Compiler provides a framework for graphically specifying an interface using state and data-frame based semantics. This specification can be transformed into a synthesizable HDL model. The disadvantage of this tool is that the user himself has to design the communication protocol.

2.3. *Protocol Conversion*

This area of research involves the design of the protocol conversion modules (aka converters, transducers) that allow the communication between the incompatible IPs. Converters are either (a) selected from a library of the pre-designed modules, or (b) generated from the description of the communicating IPs.

For example, Daveau *et al.* (1997) take a behavioural description, automatically select a protocol from a library, and generate the required interfaces to implement the communication. Passerone *et al.* (1998) describe an algorithm for generating the interfaces between HW blocks that use incompatible protocols. Given a state-based description of each communicating component, this algorithm computes the product of the two finite state machines (FSMs) that map one component protocol onto the other and vice-versa. Smith and Micheli (1998) describe the techniques for automating the process of generating interfaces between two or more components described in a HDL. Components are linked through a standard communication scheme implemented in an interface architecture that is general enough to accommodate the requirements of any domain interface. Gajski *et al.* (1998) generate buses for communication between two processes using a technique called *interface refinement*. After analysing the size of data communicated and the rate of data generation, the width of the generated bus is determined and the communication channels are merged onto the bus.

2.4. *System-Level Interface Synthesis*

The system-level interface synthesis expands the problem of the interface generation to communication between HW and SW components.

For example, Chou *et al.* (1995) present a set of algorithms for the synthesis of HW/SW interfaces. Given a functional description of component ports and detailed timming, operational, and data width information, the device drivers that enable HW/SW communication are generated. Baganne *et al*. (1997) describe a formal technique to communication synthesis for HW/SW co-design systems. A communication interface is generated at the same time as the HW module by a high-level synthesis tool. Hessel *et al.* (1999) use a multi-language design approach, which allows to transform a system composed of the subsystems, described in different languages that communicate through the communication channels using different communication primitives, into a set of interconnected processors that communicate via signals and share the communication protocols. Vahid and Tauro (1997) developed an object-oriented communication library for

HW/SW co-design. The library consists of a set of the protocol independent C/VHDL send/receive primitives, which can be used in an early design phase when the architecture is not fixed, and later refined into implementations of the selected protocols.

2.5. *Standardisation of Interfaces*

The industrial consortium VSIA has recognised the importance of simplifying the connection of system's components. So far, VSIA has proposed to standardise data formats, test methodologies, and interfaces (Cyr *et al.*, 2001). A standard VCI bus protocol can be used to connect the soft IPs with an on-chip bus. However, the competition within the IP design industry is likely to hinder the adoption of the common IP communication standards and preserve the need for synthesising interfaces between the soft IPs.

2.6. *Summary*

We summarise the related works as follows. The authors use a wide variety of the design techniques, such as graph-based ones (Madsen and Hald, 1995; Chou *et al.*, 1995), formal methods (Baganne *et al.* 1997), automata theory (Smith and Micheli, 1998; Passerone *et al.*, 1998), Petri nets (Lysecki and Vahid, 2002), grammar-based approaches (Öberg *et al.*, 1996; Siegmund and Mueller, 2000) and object-oriented techniques (Vahid and Tauro, 1997) to implement the interface synthesis. Some authors use the generative techniques (e.g., Siegmund and Mueller (2000) apply code templates to generate interface controllers).

Our approach is concerned with the generation of the IP wrappers for IP communication using the heterogeneous MPG techniques systematically. The heterogeneous MPG techniques allow us to achieve higher reusability and flexibility while developing generic components and wrapper generators. Furthermore, for us MPG is not just one of the possible ways for implementing the code generation. We use the concept of MPG as a methodological basis in our research (Štuikys *et al.*, 2002; Štuikys, 2002). In the following section, we present a framework of the heterogeneous MPG-based design methodology for the interface-based design of soft IPs .

3. A Framework of the Methodology for the Interface-based Design

3.1. *Analysis of Soft IP Interfaces*

In our analysis, we apply the principles of the *multi-dimensional separation of concerns* (Ossher and Tarr, 2000), which allow considering the orthogonal and overlapping dimensions of concerns in design space, as well as their separation, representation and integration when describing and implementing a complex system.

The soft IP design space has many dimensions, the IP interface is just one of them. The IP interface represents the external behaviour of an IP and has several levels of abstraction such as physical (logic voltage levels, current, capacity load), timing (spacing between

events), data transaction (bit-level transfer of data), packet (block-level transfer of data), and message (inter-process communication) (Borriello *et al.*, 1998). The soft IP interfaces deal only with the higher-levels of abstraction (data transaction and higher).

The main purpose of the soft IP interface is to ensure the relevant movement of data (e.g., operands, commands, addresses, values destined for a memory location, etc.) to and from the soft IP. The movement of data can be described using different rules or *protocols*, i.e., an agreed format for transmitting data between IPs.

In terms of the interface-based design, the soft IP interface plays two roles: (1) *explicitly* – to specify the I/O ports of the soft IP, and (2) *implicitly* – to specify the communication model used by the soft IP. The role (1) is achieved directly using the appropriate HDL abstractions (e.g., *entity, port* in VHDL). The role (2) requires to additionally specify the implementation details (e.g., an input signal named *CLK* or *clock* may imply that a component is driven synchronously, but the implementation of a synchronous communication model still requires a functional description).

We decompose the soft IP *interface concerns* into two dimensions as follows.

(1) The *vertical dimension*, which corresponds to the different types of I/O signals. The signals are: data signals, control signals (clock, reset, enable), communication protocol-specific signals (e.g., request, acknowledgement for handshaking), memory access signals (e.g., column select, row select, etc.), bus configuration signals (e.g., data/address multiplexing), status signals, etc.

(2) The *horizontal dimension*, which corresponds to the aspects of a particular signal, such as signal name, direction (input, output), data width and data type (e.g., *bit, std_logic* in VHDL).

In the next subsection, we describe how we use the results of the soft IP interface analysis for the interface synthesis.

3.2. *Interface Synthesis*

In order to design a HW system that is composed of the different communicating soft IPs (possibly designed by the third-party vendors), a standard (at least in terms of a designed system) communication model should be used. Initially, when a system designer selects a soft IP, he (she) usually considers only its *static* behaviour (functionality). To actually use the soft IP, however, the designer has to design or adapt its *dynamic* behaviour – the communication model (protocol). This is the issue of the interface synthesis.

We perform the interface synthesis by (1) extracting the interface parameter values from the soft IP interface (i.e., *port* statement in the VHDL *entity* description), (2) generating the modified soft IP interface, and additional functionality (glue code) to implement the communication protocol.

The soft IP *interface parameters*, which uniquely characterise the soft IP interface, are extracted from the high-level description of the soft IP interface automatically using a VHDL parser (see Fig. 1). We perform the mapping between the soft IP interface concerns and interface parameters as follows: (1) every aspect of a signal is a separate parameter, and (2) a list of the parameter values is composed of the values of aspects for every

interface signal. Further, we use the soft IP *interface parameters* to generate the modified soft IP interface needed for the glue logic.

We perform the generation of the modified interface using a set of *interface transformations* as follows. (1) *Inclusive composition* is performed when interfaces of the different soft IPs are merged together, and all signals appear in a new interface. (2) *Exclusive composition* is performed when signals with a common meaning (e.g., a shared clock) are merged during the composition of the soft IP interfaces. (3) *Type modification* is performed over selected signal(s) when type conversion is required for smooth integration of the soft IPs. (4) *Width modification* is performed over selected signal(s) when a soft IP is used in the context where the data path width is different (usually narrower). (5) *Signal inclusion* is performed when a new signal is added to the soft IP interface for control or other purposes (e.g., clock, or enable signal). (6) *Signal exclusion* is performed when a signal is removed from the soft IP interface.

To implement interface synthesis, we apply a variety of the application-specific *high-level design transformations*. These are performed on the *black-box* soft IPs. The soft IP is not actually modified, but a wrapper component(s) is (are) generated, which modify the behaviour of the soft IP *without* modifying its *internal* implementation. We consider the interface synthesis as a high-level design transformation, which performs a *superimposition* of a communication protocol on the soft IP. The result of this transformation is a component called *IP wrapper*, because it "wraps" the existing soft IP with additional functionality. In the following subsection, we consider a concept of the IP wrapper in detail.

3.3. *The Concept of IP Wrapper*

An *IP wrapper* is a component that allows adapting the interface and behaviour of another component (IP) to the requirements of its context of application. The usage of the

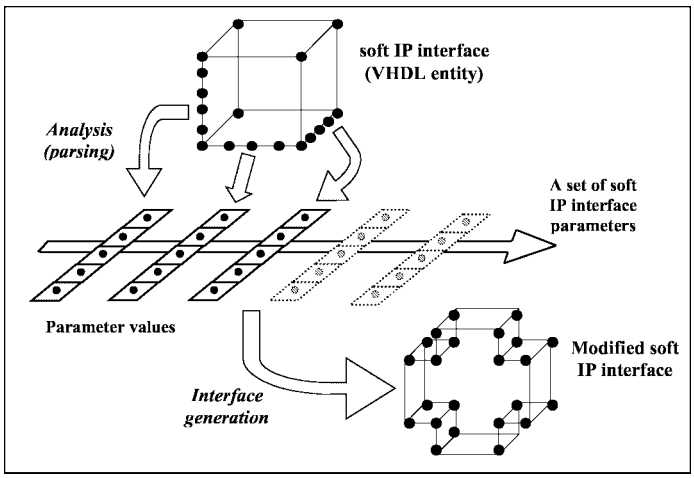

Fig. 1. Scheme of the interface synthesis.

IP wrapper (1) allows separating explicitly and implementing different design concerns separately, (2) enables the integration of the soft IPs into different HW systems, and (3) facilitates the greater reusability of the soft IPs.

We describe concisely the structure of the IP wrapper using an UML (Booch *et al.*, 1998) class diagram (see Fig. 2). The abstract class (*entity* in VHDL) *Wrapper* inherits the I/O ports of the *IP*, and declares new I/O ports for wrapper functionality. The class (*architecture* in VHDL) *IPModel* implements the functionality of entity *IP*. The architecture *WrapperModel* implements the functionality of *Wrapper* and contains component *IP*. Essentially, this description means that *WrapperModel* wraps *IPModel* with a new functionality.

Wrapping can be nested, i.e., we can apply wrappers to the wrapped soft IP again and again. Many different wrappers can be applied to the same soft IP, also the same wrapper can be applied to the different soft IPs. However, a designer must be cautious for an area and delay overhead, which is usually introduced by the wrapping procedure.

In this paper, we use heterogeneous MPG to develop the IP wrapper generator, which automatically generates the IP wrapper for any given black-box soft IP. In the next subsection, we describe how we implement the generation process using the heterogeneous MPG techniques.

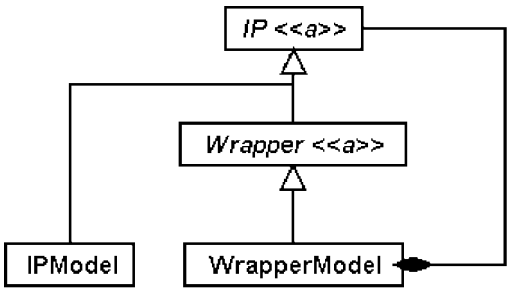

Fig. 2. Structure of the IP wrapper.

3.4. *Metaprogramming Techniques*

MPG is a higher-level programming technique, which provides means for describing generic domain programs and generating their particular instances. Two underlying principles form the basics of MPG, the *separation* of concerns and their *integration* (see Fig. 3).

To describe the essence of programming, Wirth (1976) used the following equation:

"program = data structure + algorithm".

Our equation for describing MPG is as follows (Štuikys, 2002):

"*Metaprogram = program instances as data + modification algorithm"*.

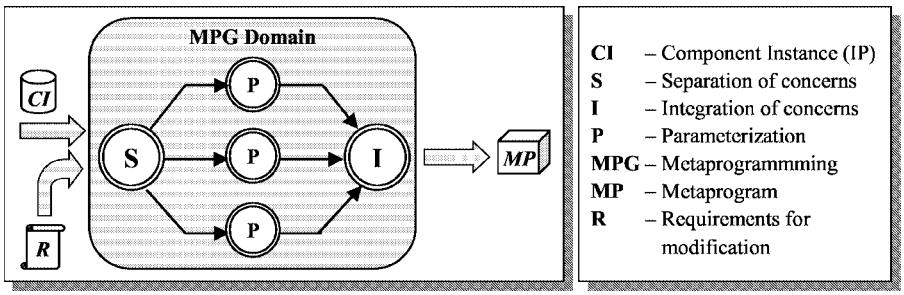

Fig. 3. Separation and integration of concerns in metaprogramming.

The main aim of MPG is to create a *metaprogram* (MP) – a generic specification, which describes explicitly the process of generation for a narrow domain of application. In fact, a MP describes a family of the related (syntactically or semantically) particular domain program instances in the context of different applications, whereas its environment (processor, compiler) generates a particular domain program instance depending upon the values of the generic parameters.

A MP is a structure, which consists of the domain program instances encapsulated with their modification algorithm. Here the modification algorithm ranges from the simple *metaconstructs* such as *metaif* (conditional generation) and *metafor* (repetitive generation) to the sophisticated application-specific *metapatterns*, which are composed of the nested combinations of the simpler metaconstructs.

The development of a MP consists of the several steps as follows. (1) The domain (usually represented by one or more available component instances and the requirements for a modification) is analysed and the modification concerns are identified and separated. These concerns represent the *variable* aspects in a domain, which are *dependable* upon the generic parameters and require the application of the MPG techniques. (2) The modification concerns are expressed through the parameters, implemented using the parameterisation-based MPG techniques, and then integrated with the *fixed* aspects of a domain, which are *orthogonal* with respect to the generic parameters. Such metaprograms describe variations in a particular programming 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 homogeneous MPG is the *implicit* separation of concerns. At the core of heterogeneous MPG is the *explicit* separation of concerns. In this paper, we consider heterogeneous MPG only.

In Fig. 4, we deliver the general framework for understanding heterogeneous MPG. Heterogeneous MPG implies the usage of two different languages. The lower-level language (*domain language*, DL) is used for expressing basic domain functionality. The higher-level language (*metalanguage*, ML) is used for expressing generalisation and describing domain code modifications. Using a ML, designer develops a MP, which is then used as a set of instructions for a ML processor to generate the specific DL code instances.

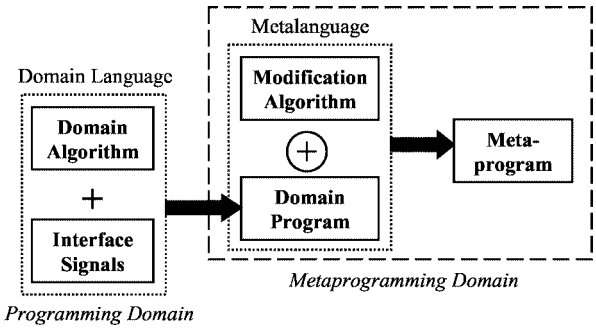

Fig. 4. Framework of the heterogeneous metaprogramming.

In the next subsection, we present an overall view to the interface-based design methodology based on MPG.

3.5. *Overall View to the Methodology*

Suppose, that a SoC designer retrieves a third-party soft IP, analyses it, and discovers that the IP needs some customisation to bridge the incompatible interfaces or to implement a different communication model in order to successfully integrate it into a particular HW system. Then the designer needs to perform the interface synthesis as follows. (1) *Extract* the IP interface parameters from the soft IP interface and use them as generic parameters for a MP. (2) *Design* a MP, where a ML performs the required DL code modifications depending upon the values of the generic parameters. (3) *Generate* the IP wrapper.

In many cases, the design process based on this methodology (see Fig. 5) can be automated for a wide variety of the application-specific architectures such as communication protocols as follows. The design step (1) is performed automatically by a DL parser, which analyses the source code of a given soft IP interface, constructs a syntax tree, and extracts the values of the interface parameters. The design step (2) is performed by the pre-designed MPs, which capture the widely used domain entities. The design step (3) is

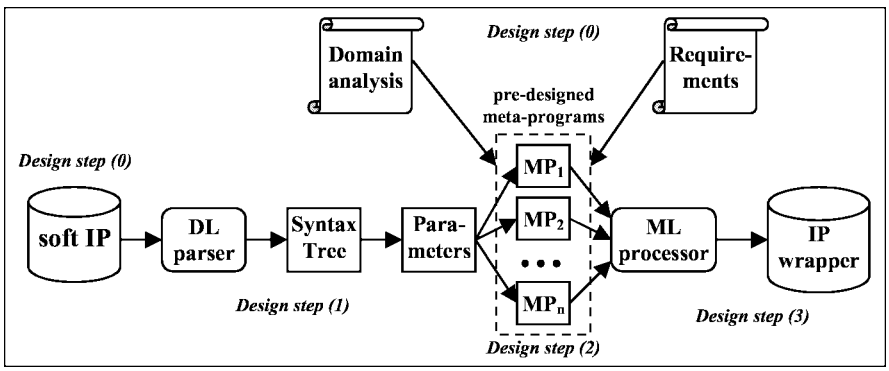

Fig. 5. Design-flow of the proposed methodology.

performed by the ML processor, which generates (instantiates) the IP wrapper. We imply that both the soft IP and MPs are reusable items, which can be designed separately and used in other applications, too. Therefore, we depict the design (or retrieval) of the soft IP and MPs as the design step (0).

We illustrate the synthesis of a FIFO communication interface for the third-party soft IPs as a case study of the proposed design methodology.

4. Case Study: Synthesis of Communication Interface

We use the concept of the IP wrapper to implement the communication interface for a synchronous FIFO protocol. The application architecture is as follows (see Fig. 6). The FIFO protocol is used in the producer-consumer communication model to smoothen bursts in the requests for a service (Gajski, 1997). The production of data by other SoC components, which are sending data to the IP, may momentarily exceed the consumption of data by the IP. To deal with such burst, the FIFOs are inserted between the data producer and the data consumer (IP). The FIFOs store the surplus data, which is read in the same order in which it was written in. The size of the FIFO determines how large a burst can be tolerated.

We treat the IP as a black-box component, of which only the interface is known. Our aim is to adapt the IP to the FIFO protocol by generating the FIFO wrapper (i.e., the IP wrapper, which implements the FIFO protocol). For this, we need to modify the IP interface and to wrap the IP with the FIFOs and control logic (FSM). Note that in this case study, we generate only a client side of the application. The wrapped IP is now ready for integration into a SoC.

We have constructed the FIFO wrapper generator to automatically generate the FIFO wrappers for the black-box third-party soft IP cores described in VHDL as follows:

(1) The VHDL parser analyses the supplied soft IP source code, constructs a syntax tree, and extracts the values of the parameters for generation. These parameters basically represent the interface signals of the soft IP.

(2) The FIFO wrapper was implemented as a set of MPs using heterogeneous MPG (Java in the role of a ML, and VHDL as a DL). Each MP is a Java class, which encapsu-

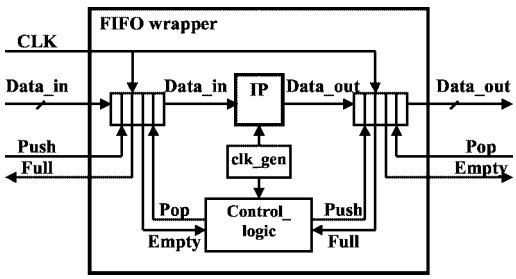

Fig. 6. Architecture of the FIFO wrapper.

Table 1
Synthesis results

IP	Area, cells (IP)	Area, cells (FIFO (4))	Increase	Est. power, uW (IP)	Est. power, uW (FIFO (4))	Increase
Free-6502	4670	6880	47%	8.2693	13.2107	37%
Dragonfly	5883	10451	44%	19.9421	25.3074	21%
DLX	20118	25735	22%	23.2075	28.3474	18%

lates a generic domain entity (e.g., FIFO, FSM, etc.) and generates a specific instance of it in VHDL according to the values of the generic parameters specified via the constructor.

(3) The FIFO wrapper generator performs wrapping of the soft IP by generating the instances of the FIFO buffer, a *port map* statement to map the signals of the FIFO wrapper to the soft IP, and additional control logic (FSM), which handles the data flow between the input FIFO, the soft IP, and the output FIFO. The wrapper interface is synthesised using the *exclusive composition* technique (see Section 3.2). For the asynchronous soft IPs, the *signal inclusion* is used to insert a clock signal.

We have performed the experiments using the third-party soft IPs as follows: (1) Free-6502 8-bit micro-processor (Kessner, 1999); (2) DRAGONFLY 8-bit core (LEOX, 2001); (3) DLX 32-bit micro-processor (Gumm, 1995). Synthesis results (we use Synopsys tools; CMOS 0.35 um technology) for the original soft IPs and the generated components (FIFO size is 4) are presented in Table 1.

The synthesis results show an average increase of about 38% in chip area (the ratio tends to decrease as the area of the soft IP increases), and about 25% in estimated power usage for the generated FIFO(4) wrappers.

5. Evaluation and Discussion

The complexity in SoC design can be managed at the relevant extent by raising the level of abstraction through the introduction of the new design methodologies, which deal with reuse and integration of the pre-designed soft IPs. As the IP-based approaches predominate in SoC design now, the role of the high-level design processes, such as wrapping for interface synthesis, grows increasingly.

A higher abstraction level in HW design can be achieved either by extending the capabilities of the standard HDL languages, or applying the multi-language design paradigm. Metaprogramming (MPG) is a specific case of that paradigm. In this paper, we considered the usage of *heterogeneous* MPG, which is performed using an external metalanguage (ML). The aim the ML is to deliver the means for describing modifications of a domain program and generating its particular instance written in a domain language (DL). So, heterogeneous MPG implements the multi-language design paradigm, and allows achieving a clearer separation of concerns in a design.

The usage of MPG for the interface-based design allows us to develop metaprograms (soft IP wrapper generators), which (1) encapsulate common domain solutions (com-

munication protocols), and (2) allow flexible adaptation of the soft IPs to the different communication models.

We also have applied UML to describe the structure of the IP wrapper at a high level of abstraction. The benefits of using UML in HW design are as follows. (1) The level of abstraction is raised, which allows dealing with growing complexity of the HW designs. (2) The comprehensibility of the design process is increased. (3) The design productivity can be significantly increased, because the automated code generation can be implemented.

6. Conclusions and Future Work

We have presented an approach to the interface-based design, which is based on the systematic usage of the heterogeneous metaprogramming (MPG) techniques. MPG allows us (1) to raise the level of abstraction, and (2) to increase the reusability and customisability of the soft IPs, and (3) to generate automatically the communication interfaces. In order to adapt the soft IPs to the context of application, we have used the concept of IP wrapper, which allows the explicit separation of different design concerns, and enable the integration of the soft IPs into different HW systems.

Future work will focus on the creation of a library of the generators for the communication interface synthesis using different data protocols.

Acknowledgements

We would like to thank the anonymous referees for their insightful comments and suggestions for improving the paper.

References

Baganne, A., J.L. Philippe and E. Martin (1997). A formal technique for hardware interface design. In *Proc. of IEEE International Symposium on Circuit and Systems (ISCAS 97)*, Vol. 3. pp. 1592–1595.

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

Borriello, G., L. Lavagno and R.B. Ortega (1998). Interface synthesis: a vertical slice from digital logic to software components. In *Proc. of International Conference on Computer-Aided Design (ICCAD 1998)*, San Jose, CA, USA. pp. 693–695.

Chou, P.H., R.B. Ortega and G. Borriello (1995). Interface co-synthesis techniques for embedded systems. In *Proc. of the IEEE/ACM International Conference on Computer Aided Design (ICCAD 95)*, San Jose, CA, USA. pp. 280–287.

Cyr, G., G. Bois and M. Aboulhamid (2001). Synthesis of communication interface for SoC using VSIA recommendations. In *Proc. of DATE Design Forum 2001*, Munich, Germany. pp. 155–159.

Daveau, J.M., G.Marchioro, T. Ben–Ismail and A.A. Jerraya (1997). Protocol selection and interface generation for Hw–Sw co-design. In *IEEE Transactions on VLSI Systems*, Vol. 5, No. 1. pp. 136–144.

Gajski, D. (1997). *Principles of Digital Design*. Prentice Hall.

Gajski, D.D., F. Vahid, S. Narayan and J. Gong (1998). SpecSyn: an environment supporting the specify-explore-refine paradigm for hardware/software system design. *IEEE Transactions on Very Large Scale Integration Systems*, Vol. 6, No. 1. pp. 84–100.

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

Gutberlet, P., and W. Rosenstiel (1994). Specification of interface components for synchronous data paths. In *Proc. of the 7th International Symposium on System Synthesis (ISSS'94)*. pp. 134–139.

Hessel, F., P. Coste, P. LeMarrec, N. Zergainoh, J. Daveau and A. Jerraya (1999). Communication interface synthesis for multilanguage specifications. In *Proc. of the 10th IEEE International Workshop on Rapid System Prototyping (RSP 1999)*, Clearwater, Florida, USA. pp. 15–20.

Keating, M., and P. Bricaud (1999). *Reuse Methodology Manual for System-on-a-chip Designs*. Kluwer Academic Publishers.

Kessner, D. (1999). Free-6502 core. http://www.free-ip.com/6502/

Lennard, C.K., P. Schaumont, G. de Jong, A. Haverinen and P. Hardee (2000). Standard for system-level design: practical reality or solution in search of a question? In *Proc. of Design, Automation and Test in Europe Conference (DATE 2000)*, Paris, France. pp. 576–583.

LEOX Team (2001). DRAGONFLY micro-core. http://www.leox.org

Lysecki, R., and F. Vahid (2002). Prefetching for improved bus wrapper performance in cores. *ACM Transactions on Design Automation for Electronic Systems*, Vol. 7, No. 1. pp. 58–90.

Madsen, J., and B. Hald (1995). An approach to interface synthesis. In *Proc. of the 8th International Symposium on System Synthesis (ISSS'95)*, Cannes, Cote d'Azur, France. pp. 16–21.

Öberg, J., A. Kumar and A. Hemani (1996). Grammar-based hardware synthesis of data communication protocols. In *Proc. of the 9th International Symposium on System Synthesis (ISSS'96)*, San Diego, CA, USA. pp. 14–19.

Ortega, R., and G. Borriello (1998). Communication synthesis for distributed embedded systems. In *Proc. of International Conference on Computer Aided Design (ICCAD 98)*, San Jose, CA, USA. pp. 437–444.

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

Passerone, R., J. Rowson and A. Sangiovanni–Vincentelli (1998). Automatic synthesis of interface between incompatible protocols. In *Proc. of the 35th Design Automation Conference (DAC98)*, San Francisco, CA, USA. pp. 8–13.

Rajawat, A., M. Balakrishnan and A. Kumar (2000). Interface synthesis: issues and approaches. In *Proc. of the 13th International Conference on VLSI Design*, Calcutta, India. pp. 92–97.

Rowson, J., and A. Sangiovanni–Vincentelli (1997). Interface-based design. In *Proc. of the 34th Design Automation Conference (DAC 97)*, Anaheim, CA, USA. pp. 178–83.

Seawright, A., U. Holtmann, W. Meyer, B. Pangrle, R. Verbrugghe and J. Buck (1996). A system for compiling and debugging structured data processing controllers. In *Proc. of European Conference on Design Automation (EuroDAC96)*, Geneva, Switzerland. pp. 86–91.

Sgroi, M., M. Sheets, A. Mihal, K. Keutzer, S. Malik, J. Rabaey and A. Sangiovanni–Vincentelli (2001). Addressing the system-on-a-chip interconnect woes through communication-based design. In *Proc. of the 38th Design Automation Conference (DAC 2001)*, Las Vegas, Nevada, USA. pp. 667–672.

Sheard, T. (2001). Accomplishments and research challenges in metaprogramming. In *2nd International Workshop on Semantics, Application, and Implementation of Program Generation (SAIG'2001)*, Florence, Italy. *Lecture Notes in Computer Science*, **2196**, pp. 2–44.

Siegmund, R., and D. Mueller (2000). A method for interface customization of soft IP cores. In R. Seepold and M. Navidad (Eds.), *Virtual Component Design and Reuse*. Kluwer Academic Publishers.

Smith, J., and G.D. Micheli (1998). Automated composition of hardware components. In *Proc. of the 35th Design Automation Conference (DAC98)*, San Francisco, CA, USA. pp. 14–19.

Štuikys, V., R. Damaševičius, G. Ziberkas and G. Majauskas (2002). 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. pp. 257–266.

Štuikys, V. (2002). Metaprogramming techniques for program generation and soft ip design. *Research Report Presented for Habilitation*, KTU, Technologija.

Vahid, F., and L. Tauro (1997). Object-oriented communication library for hardware-software codesign. In *Proc. of the 5th International Workshop on Hardware/Software Codesign (CODES/CASHE'97)*, Braunschweig, Germany. pp. 81–86.

Wirth, N. (1976). *Algorithms + Data Structures = Programs*. Prentice–Hall, Englewood Cliffs, New Jersey.

R. Damaševičius received MSc. degree in informatics from Kaunas University of Technology, Lithuania in 2001. Currently he is PhD student at Informatics faculty, 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.

V. Štuikys received PhD degree from Kaunas Polytechnic Institute in 1970 and Dr. habil. from Kaunas Polytechnic University of Technology in 2003. Currently he is a professor at Software Engineering Department, Kaunas University of Technology, Lithuania. His research interests include domain-specific and software reuse, high level domain-specific languages, component-based programming, metaprogramming and program generation, expert systems and CAD systems, and soft IP design.

Lanksčių intelektualiosios nuosavybės komponentų komunikavimo sąsajų kūrimas naudojant heterogeninį metaprogramavimą

Robertas DAMAŠEVIČIUS, Vytautas ŠTUIKYS

Šiame straipsnyje mes aptarėme heterogeninio metaprogramavimo taikymą, kaip kurti komunikavimo sąsajas tarp lanksčių Intelektualiosios Nuosavybės (IN) komponentų. Heterogeninis metaprogramavimas remiasi dviejų kalbų – srities ir metakalbos – naudojimu toje pačioje specifikacijoje. Srities kalba yra naudojama srities funkcionalumui aprašyti, o metakalba – bendriniams komponentams ir programų generatoriams kurti. Mes pateikėme projektavimo metodologiją, pagrįstą metaprogramavimo metodais. Mūsų metodo naujumas yra sisteminis heterogeninio metaprogramavimo taikymas kuriant IN įvyniojimo komponentus, kurie yra skirti komunikavimui tarp trečios šalies lanksčių IN komponentų. Mes pateikėme eksperimentą sukurdami komunikavimo sąsają, kuri naudoja FIFO protokolą.