Evaluating UML2 Modeling of IP-XACT Objects for Automatic MP-SoC Integration onto FPGA

Tero Arpinen, Tapio Koskinen, Erno Salminen, Timo D. Hämäläinen, Marko Hännikäinen
Tampere University of Technology,
Department of Computer Systems,
P.O. Box 553, FI-33101 Tampere, Finland
Email: tero.arpinen@tut.fi

***Abstract*—IP-XACT is a standard for describing intellectual property metadata for System-on-Chip (SoC) integration. Recently researchers have proposed visualizing and abstracting IP-XACT objects using structural UML2 model elements and diagrams. Despite the number of proposals at conceptual level, experiences on utilizing this representation in practical SoC development environments are very limited. This paper presents how UML2 models of IP-XACT features can be utilized to efficiently design and implement a multiprocessor SoC prototype on FPGA. The main contribution of this paper is the experimental development of a multiprocessor platform on FPGA using UML2 design capture, IP-XACT compatible components, and design automation tools. In addition, modeling concepts are improved from earlier work for the utilized integration methodology.**

I. Introduction

The complexity of embedded systems requires advanced methods to compose System-on-Chips (SoC). Reuse of Intellectual Property (IP) components and design automation must be effective in SoC development. Moreover, a typical SoC development environment includes several vendors, tools, and implementation languages. IP-XACT [1] standard was developed by SPIRIT consortium to ease the composition of reusable IPs from multiple vendors. IP-XACT defines IP metadata XML schema and design tool interfaces for SoC integration.

Unified Modeling Language (UML) version 2 [2] is an object-oriented graphical language for Model Driven Development (MDD). Traditionally, UML has been used for visualizing, specifying, constructing, and documenting artifacts of software-intensive systems [3], but currently it is emerging in the embedded domain as well. Its adoption is seen promising for specification of requirements, behavioral and architectural modeling, test bench generation, and IP integration [4].

Recently researchers have proposed visualizing certain IP-XACT objects using structural UML2 model elements and diagrams together with UML profiles [5] [6] [7]. Despite the number of proposals, experiences on utilizing such representation in practical SoC development environments are very limited.

This paper presents how UML2 modeling of IP-XACT features can be used to efficiently design and implement a multiprocessor SoC platform on FPGA. The main contribution of this paper is the experimental prototype development on FPGA using the methodology and tools presented in [7]. The prototype platform contains multiple processing elements, an on-chip communication network, and interfaces to several off-chip devices. Moreover, the modeling concepts are improved from [7] to further assist design capture and library documentation. The utilized modeling concepts are upgraded so that they are applicable for both IP-XACT versions 1.2 and 1.4.

The rest of the paper is structured as follows. Section II gives an introduction to central concepts of IP-XACT. Section III examines the related work. Section IV presents the utilized design methodology and automation tools. Section V presents the utilized modeling concepts. The modeling and implementing the prototype platform is presented in Section VI. The experiments performed during the prototype development are presented in Section VII. Section VIII concludes the paper.

II. SPIRIT IP-XACT

The SPIRIT IP-XACT v1.4 schema specifies the metadata structure for describing IP, their composition and configuration in SoC design. There are seven top-level schema objects in IP-XACT: *IP-XACT component*, *bus definition*, *abstraction definition*, *design*, *design configuration*, *abstractor*, and *generator chain*. The first four of them are considered in our work and shortly introduced next.

Component is the central placeholder for metadata of a single HW IP, such as a processor, peripheral, or bus. Components may reference designs to create a hierarchy. The component metadata includes bus interfaces, physical signals, model parameters, source file sets, address spaces, and generators associated with the component.

Bus definition together with associated *abstraction definitions* specify an interface between components. In this paper, this combination is referred as *interface definition*. Abstraction definition defines a set of ports, which are classified into wire ports and transactional ports. Wire ports carry logic values or arrays of logic values. They are used to specify logical names, widths, and directions of register transfer level (RTL) signals. Transactional ports carry complex information modeled at high level of abstraction, such as transaction-level model (TLM). Bus and abstraction definitions are referenced by bus interfaces in component descriptions.

978-3-9810801-5-5/DATE09 © 2009 EDAA

Design describes the assembly and configuration of components in a HW system or subsystem. It describes component instances and interconnections between them. Moreover, it may describe the internal structure of a hierarchical component. Internal bus interfaces can be exported out of the design so that they become accessible in the upper level of hierarchy. Furthermore, design is used to set model parameter values for component instances.

III. Related work

UML2 offers natively a support for modeling computational resources in form of nodes and communication paths represented in deployment diagrams. However, the expression power of deployment diagrams is too limited for the needs of modeling heterogeneous embedded HW structures. The biggest limitation is the lack of notion for HW interfaces. UML2 has also other structural metaclasses and diagrams that have potential for detailed HW modeling, such as classes, interfaces, parts, ports and connectors. In addition, UML2 has a profiling mechanism, which allows the creation of new model elements for a specific domain by specializing UML2 metaclasses with stereotypes, tag definitions and constraints.

A. HW resource modeling with UML profiles

There are several UML profiles that support embedded HW resource modeling. In the following, two of the closest to related research are examined.

The UML profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) [8] is a proposal for OMG standard profile. It is a promising extension to modeling concepts that play key roles in real-time embedded systems development. These include for example, quantifiable notion of time, non-functional properties, execution platform, and allocation (or mapping) of functionality. The profile is methodology-independent and it offers a common set of standard notations and semantics for a designer to choose from while still allowing addition of custom extensions. Using this kind of profile may lead to a number of specialized profile instances, each dedicated for their own methodology. Nevertheless, using a standard profile in general promotes commonly known notations and semantics between designers.

TUT-Profile [9] is a proprietary profile for embedded systems development. It is specialized to be used as a front-end for an MP-SoC design framework called Koski [10]. TUT-Profile has been tailored for model-based design-space exploration, application code generation, and IP integration. Due to its specialized nature, TUT-Profile can be considered as methodology-dependent. The benefit of this specialization is the concise set of resulting modeling concepts, that is, model elements are defined only if they are needed to implement the methodology. On the other hand, this comes with a price of lack of standardization.

B. IP-XACT modeling in UML

This paper considers three proposals for modeling IP-XACT objects using UML2 language. In the following, a short introduction to each of them is given.

Andre *et. al* [5] utilize the MARTE profile as the basis of their work. In their proposal, they specialize MARTE profile for IP-XACT v1.4 by first creating a metamodel for several IP-XACT objects and then mapping these to UML2 metaclasses. The proposal introduces several mapping rules between MARTE model elements and IP-XACT objects. However, the introduced concepts have not been utilized, to our knowledge, in a real SoC development environment.

Schattkowsky *et. al* [6] present a proprietary UML2 profile for IP-XACT v1.4 objects. The modeling concepts are reported mainly in form of illustrative examples. The proposed profile is utilized in practice in a tool framework that enables generation of SystemC transaction-level model (TLM). Integration of RTL components is not reported.

In our earlier work [7] we proposed mapping of stereotyped UML2 elements and main IP-XACT v1.2 objects using TUT-Profile. The mapping rules covered UML2 abstraction of IP-XACT components, hierarchical components, designs, bus definitions, interfaces as well as configuring model parameters of components. We also presented an associated tool flow that allows automatic RTL component integration based on the proposed transformation rules. Details of the prototype platform and corresponding example models were minor.

All proposals allow representing IP-XACT designs, components, interface definitions, and bus interfaces with slight variations. Each proposal effectively exploits structural UML2 metaclasses and diagrams for presentation. Main differences are the utilized stereotype names, interface representations, and number of modeled objects.

Despite the large number of proposals, a detailed case study of a complex MP-SoC implementation using UML2 and IP-XACT based methodology has not been reported. Thus, in this paper we present a design and implementation of a multiprocessor system on FPGA using the methodology and tool framework presented earlier in [7]. The same UML2 metaclasses and diagrams are used for representing the main IP-XACT objects as in our earlier work with TUT-Profile. In addition, new modeling concepts are introduced in this work to further assist design capture and model library management.

C. Feasible modeling notations for IP-XACT

Model is a simplification of reality and *every system is best approached through a small set of nearly independent models* [3]. Creating understandable models with UML profiling mechanism requires that defined stereotype extensions and tagged values truly represent the modeled concepts in reality. Following common notations of an existing modeling language should not make model descriptions less understandable nor bias the formulation of the original modeling problem or utilized methodology. Using mapping rules between IP-XACT objects and predefined model elements of profiles that have been originally targeted to some other modeling problems, such as in [5] and [7], may lead to complicated and confusing model representation. In the context of representing IP-XACT metadata objects with stereotyped UML model elements, we conclude that

1) UML2 structural metaclasses and diagrams are well-suited for graphically representing relations of main IP-XACT objects.
2) The most efficient and designer-friendly way is to use concise and proprietary stereotype extensions tailored specifically for IP-XACT concepts and utilized design methodology.

Thus, in this paper we keep the same UML metaclasses for representing main IP-XACT objects as earlier with TUT-Profile [7], but the stereotype extensions used are specifically tailored for IP-XACT concepts as it makes the semantics of the models more understandable.

IV. Utilized design methodology and tools

The design methodology adopted in this paper is presented in Fig. 1. The flow begins from capturing IP-block level design of the system HW using UML2 description. This is transformed into IP-XACT design description in the model parsing phase. Component and interface definition descriptions are predefined and contained in an IP library. They have corresponding abstract models in the UML2 model library. The IP-XACT descriptions are used for generating the top-level structure in RTL HDL. Finally, the system is synthesized onto FPGA.

The utilized subtools implementing the methodology are UML2 modeling tool, model parser, VHDL generator, compilation environment configurator, and synthesis tool for FPGA. Commercial Tau G2 tool from Telelogic is used for UML2 modeling. In addition to IP-block level design capture, it is used to define the required stereotype extensions and a model library for IP components and interface definitions. The model parser converts the created UML2 model into IP-XACT design description. The VHDL generator creates a top-level structural VHDL description based on the IP-XACT metadata descriptions. The compilation environment configurator is a tool that detects the signals that are exported out of the design (from IP metadata) and assigns them to FPGA pins. It generates a tool-specific description file which contains mapping between top-level design signal names and FPGA pin labels. This file is forwarded to Altera Quartus II tool which performs HW synthesis for FPGA. Table I shows implementation language and source code lines of each self-made tool.

V. Modeling IP-XACT objects in UML2

In general, the selection of the IP-XACT concepts represented in UML2 is dependent on the modeling purpose and methodology. Representing all IP-XACT objects with UML2 notations is clearly not desired for our, if any, methodology. Instead, the goal is to allow system-level design capture with abstract UML2 models while the low-level component integration details (such as wire names, widths, and directions) are kept in IP library as IP-XACT metadata. This section defines the selection of UML2 model elements for IP-XACT objects that are necessary and feasible for capturing the IP-block level

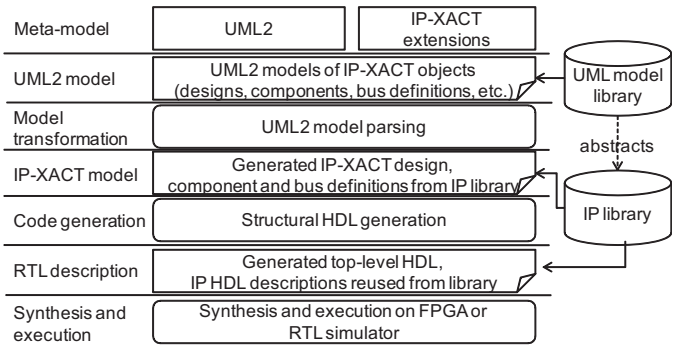

Fig. 1. Utilized methodology for automatic HW platform integration.

TABLE I
Tool implementation details.

Tool	Language	Code lines
Model parser	TCL / Python	502 / 1077
VHDL generator	Python	2149
Compilation env. conf.	Python	384

design in UML2 and representing IP-XACT component library with corresponding UML2 model library.

IP-block level design capture in UML2 requires that objects of IP-XACT design have corresponding UML2 representation. These objects include component instance, bus interface, interconnection, and configurable element. Configurable element is a convenient concept for forwarding configuration data to the HDL generator, which maps the data to instance-specific parameters or generics at HDL code level. Model elements for IP-XACT component, bus definition, and abstraction definition are also needed. This is neccessary for two reasons. First, to construct the UML2 model library from components which can be conveniently instantiated. Second, to bind utilized bus interfaces with interface definitions. Bus interfaces and model parameters of a component are also neccessary to indicate how components can be interconnected and configured, respectively, in a design. Information concerning individual wire ports of interfaces are not needed in constructing a design and thus they can be abstracted from the UML model.

Moreover, a mechanism for associating UML2 model library elements with IP-XACT objects in the IP library has to be provided. IP-XACT schema defines an element called VLNV (Vendor, Library, Name, Version), which is suitable for this purpose.

Fig. 2 shows the selected stereotype extensions to represent IP-XACT features for the utilized design methodology. IP-XACT component corresponds to a class stereotyped as *Component*. Bus interface is modeled as a stereotyped UML port owned by a *Component* class. This port requires or provides a single UML interface, which is given the properties of a bus definition and an abstraction definition using correspondingly named stereotypes. The reason for this kind of interface representation comes from the analogous nature between UML port and IP-XACT bus interface (end points of communication, *owned* by communicating entities) as well as analogous nature between UML interface and IP-XACT bus definition (common

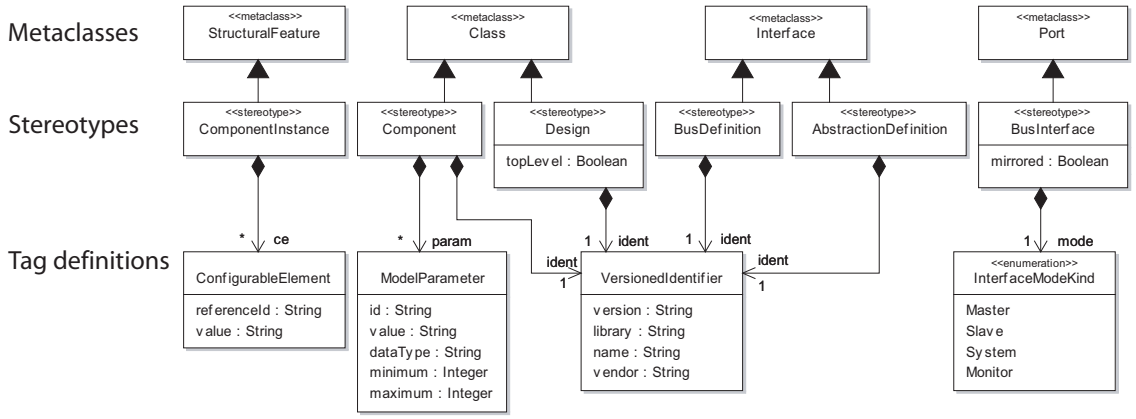

Fig. 2. Stereotype extensions for IP-XACT.

contracts for communication, *implemented* by the end points of communicating entities).

Internal structure of a UML class with parts and connected ports are used to model IP-XACT design with component instances and interconnections, respectively. This can be graphically presented in a composite structure diagram. *Design* stereotype is applied to a class that represents an IP-XACT design as its internal structure. This stereotype has an attribute *topLevel:Boolean* that defines if the stereotyped class represents the top-level structure of the HW architecture. This is an indication for model parsing tool where to start parsing the design hierarchy which may contain multiple levels. A hierarchical component is modeled by applying both *Component* and *Design* stereotypes for the same class.

To associate UML2 model elements with real IP-XACT object descriptions, the VLNV element is represented as a data type named *VersionedIdentifier*. Model parameters are represented using a data type named *ModelParameter* and extending the *Component* stereotype with a tagged value list *param:ModelParameter[*]*. Instance-specific parameter values are set using tagged values list *ce:ConfigurableElement[*]* of the *ComponentInstance* stereotype.

VI. Modeling and implementing the prototype MP-SoC platform onto FPGA

The prototype platform instances are developed onto Altera Stratix II EP2S180 DSP development board. It contains a Stratix II FPGA and several off-chip memories, I/O devices and peripherals that are connected to the FPGA circuit. The desired platform architecture on FPGA is presented in Fig. 3. The platform is composed of several Nios II softcore CPU subsystems, an SDRAM controller, and two accelerators for dedicated video encoding functions. The components are interconnected by Heterogeneous IP Block Interconnection (HIBI) on-chip communication architecture. Nearly similar kind of architecture has been used for implementing parallel MPEG-4 video encoding systems [11].

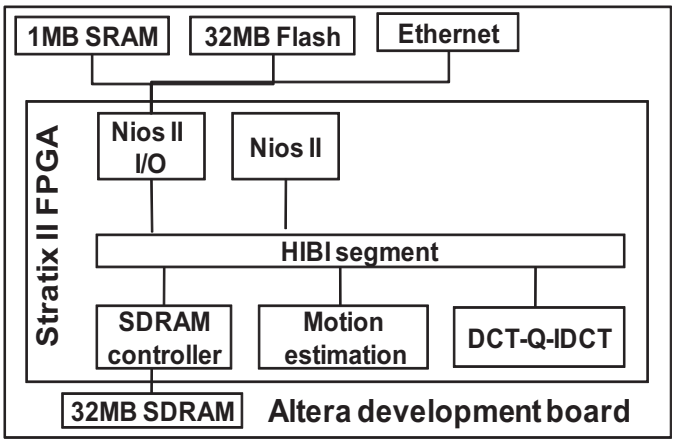

Fig. 3. Platform architecture on Altera FPGA development board.

A. Preparing platform IPs for integration

Altera provides SOPC builder tool [12] to configure Nios II based systems from a library of IPs and generate required Avalon interconnect logic. We use this tool in collaboration with our framework to create reusable subsystems containing a single Nios II core connected to a timer, JTAG UART module, program memory, and an interface logic block to connect to an HIBI segment. In practice, the number of possible Nios II CPU subsystem variations is enormous and each of these would have to have their own IP-XACT component description. An efficient solution with these kinds of configuration tools would be to automatically produce the corresponding IP-XACT component description. Currently, our solution is to generate different Nios II processor subsystems and write IP-XACT descriptions for them manually. The subsystems are placed into our IP library for reuse. It should be emphasized that self-made tools in our framework, excluding the model parser, are not tightly-coupled with the SOPC builder, TAU G2, or any other third-party tool as information exchange is carried out in form of IP-XACT descriptions.

A subsystem with Nios II *fast* core and the above mentioned peripherals is created. In addition, a specialized CPU subsys-

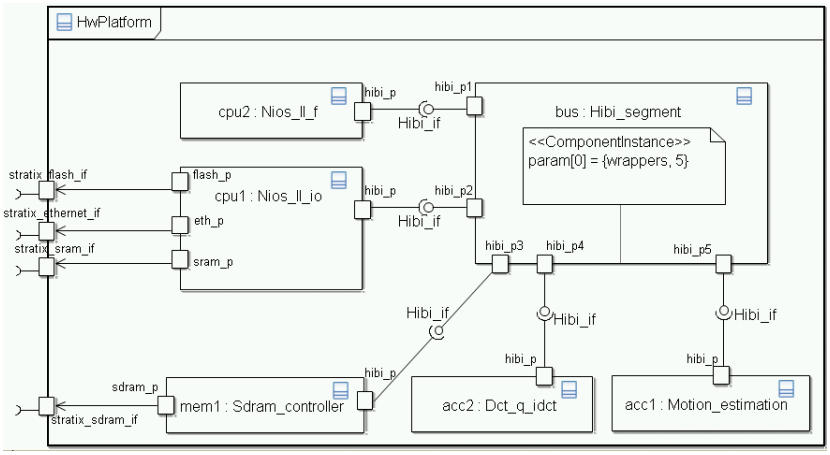

Fig. 4. View of platform instance model in a UML2 composite structure diagram.

tem is created that has an access to the external SRAM, Flash, and Ethernet controller of the FPGA development board. This specialized processor subsystem is named Nios II I/O. Dedicated hardware blocks for SDRAM controller and video encoding functions are custom made and each contain an HIBI interface. The SDRAM controller has also dedicated interface signals to the off-chip SDRAM chip.

For the prototype platform, a single HIBI segment component containing up to 21 bus wrappers is created. The realized number of bus wrappers is configured at RTL level using a VHDL generic. The number of wrappers is a model parameter for the component. As the HIBI interconnection architecture is symmetrical in nature, all HIBI bus interfaces provided by its ports are *mirrored* master interfaces. Respectively, all other IPs connecting to HIBI segment own a port that requires an HIBI master interface. IP-XACT descriptions for all platform IPs are created manually.

B. Modeling IP library components in UML

When a new IP with an associated IP-XACT description is developed or purchased from a third party vendor, it has to be properly imported before it can be used within the framework. In practice, this is carried out by adding the IP package as a whole into the IP library and creating a new model for it in the UML model library.

The phases in adding a new component to the model library are as follows

1) Creating a new class element to the model library.
2) Stereotyping the class as *Component*.
3) Defining the *VersionedIdentifier* and *ModelParameter* tagged values for the component model according to IP-XACT description.
4) Adding a port stereotyped as *BusInterface* for each bus interface in the IP-XACT description. Selecting the interface definition (UML interface) and defining the interface mode (with tagged values) for each bus interface port.

Addition of a new component to the model library typically takes only a couple of minutes with the UML modeling tool. Adding a new interface definition also requires actions for the model library. It consists of creating a UML interface, stereotyping it as bus definition and abstraction definition, and defining *VersionedIdentifier* tagged values for both definitions corresponding to their IP-XACT descriptions.

Models for all of our platform component and interface definitions are created by following these steps.

C. Modeling the platform design in UML

Modeling the platform design in UML2 is in practice realized in the UML modeling tool by instantiating component classes from model library, interconnecting them together, and setting their parameters. Fig. 4 shows how our platform components are instantiated as parts and connected using ports and connectors in a composite structure diagram. The model parameter value for HIBI segment component instance is set as a configurable element. The boundaries of the composite structure diagram practically correspond to FPGA chip boundaries. Bus interfaces that are defined by the ports of the HW platform become signals that are exported out of the top-level entity and FPGA I/O-pins. Ad-hoc signals for clock and reset are manually added to the generated IP-XACT design.

VII. Experiments on the case study

The benefits in development times were tested by composing the top-level structure of a platform instance with two Nios II CPU subsystems using three different approaches. The first approach was to write the top-level VHDL file according to a given specification without any automation. The second approach was to manually write the IP-XACT design for the platform and automatically generate VHDL. The third approach was to describe the platform in UML2 and fully utilize the automated tool flow. After synthesis for FPGA, test programs were successfully executed on platform instances. Each approach was performed once by an experienced designer.

The measured top-level VHDL development times for the case study using the different approaches are presented in Table II. The difference between modeling the architecture

TABLE II
DESIGN CAPTURE TIMES.

Type of design capture	Time used
Manually writing VHDL	**1 h 25 min**
Manually writing IP-XACT design	**26 min**
Creating a UML model	16 min
+ modifications to IP-XACT design	4 min
Total	**20 min**

TABLE III
TOOL EXECUTION TIMES.

UML model parser	19-37 s
VHDL generator	11-30 s
Compilation environment configurator	16-36 s

in UML and manually writing the IP-XACT design was six minutes. UML modeler had to make some manual modifications to the generated IP-XACT design to make the system functional. These included the addition of ad-hoc connections for clock and reset signals.

It should be also noted that this evaluation does not take into account the initial time costs to set up the design environment. These include the creation of UML component models (for the third approach) and IP-XACT descriptions (for the second and third approaches). However, these time costs occur only once.

Tool execution times for the case study platform are presented in Table III. Measurements were run on a Pentium 4 HT 3.2 GHz with 1 GiB of memory. In addition to times presented on the table, HW synthesis took 34 minutes. It was done with Altera Quartus version 7.2. The execution times of the self-made tools that create the top-level VHDL description are very short in comparison to overall design time.

Table IV presents the number of metadata files and description lines used for the prototype platform. This illustrates the effort of creating metadata descriptions for our platform components. The amount of lines in component definitions is largely affected by the component definition of the HIBI segment that has 7134 lines. This is due to the fact that the HIBI segment component has been designed to offer 21 agent interfaces altogether. A single pin map file is needed for each interface that is exported out of the design.

The number of generated code and description lines for the prototype platform are presented in Table V. The amount of VHDL code lines compared to IP-XACT design description demonstrate the higher abstraction of IP-XACT design.

A. Discussion on results

The measured values in the prototype development times and code lines suggest that the utilization of IP-XACT and automation brings a significant advantage in integration efficiency when IP-XACT component and interface definition descriptions are reused or provided by IP vendors. IP-XACT design capture using UML2 notations and automatic IP-XACT design XML generation can save some time when constructing or modifying IP-XACT designs. The UML2 approach also helps in visualizing and documenting IP-XACT designs in

TABLE IV
MANUALLY CREATED METADATA FILES.

Metadata type	Number of files	Number of lines
Bus definition	6	560
Component definition	6	10,074
Pin map	4	222

TABLE V
GENERATED FILES.

Generated file	Generated by	Number of lines
IP-XACT Design	Model parser	109
Top-level VHDL	VHDL generator	902
Compilation project settings	Compilation env. configurator	270

form of notations and diagrams that are based on a standard modeling language. Furthermore, IP-XACT XML files contain massive amount of information on integration details. Representing components and their main IP-XACT properties in UML2 helps in documenting the IP-XACT library and highlighting the component properties that are required for design description.

VIII. CONCLUSIONS

This paper presented how UML2 modeling of IP-XACT features can be utilized to efficiently design and implement a multiprocessor SoC prototype on FPGA. Our main contribution was the experimental development of an MP-SoC platform on FPGA using a UML2 and IP-XACT integration methodology and tool framework. Modeling concepts were improved for the particular methodology from earlier work. Experiments in the prototype development suggest that IP-XACT and automation can significantly improve efficiency of integration when component IP-XACT descriptions are reused or provided by IP vendors. Furthermore, UML2 description improves visualization and documentation of IP-XACT designs and libraries without excessive extra effort.

REFERENCES

[1] *IP-XACT v1.4: A specification for XML meta-data and tool interfaces*, Schema Working Group of The SPIRIT Consortium, March 2008.

[2] Object Management Group (OMG), *OMG Unified Modeling Language (OMG UML) Superstructure, V2.1.2*, Nov. 2007.

[3] G. Booch, J. Rumbaugh, and I. Jacobson, Eds., *The Unified Modeling Language User Guide*. Addison-Wesley Professional, 1998.

[4] G. Martin and W. Mueller, Eds., *UML for SOC Design*. Springer, May 2005.

[5] C. Andre *et al.*, "Modeling SPIRIT IP-XACT with UML MARTE," in *DATE'08*, March 2008.

[6] T. Schattkowsky and T. Xie, "UML and IP-XACT for Integrated SPRINT IP Management," in *UML-SOC'08*, June 2008.

[7] T. Arpinen *et al.*, "Model-driven Approach for Automatic SPIRIT IP Integration," in *UML-SOC'08*, June 2008.

[8] Object Management Group, *A UML Profile for MARTE, Beta 1 Specification*, August 2007.

[9] T. Arpinen *et al.*, "Modeling Embedded Software Platforms with a UML Profile," in *FDL'07*, April 2007.

[10] T. Kangas *et al.*, "UML-Based Multi-Processor SoC Design Framework," *ACM TECS*, vol. 5, no. 2, pp. 281–320, May 2006.

[11] A. Rasmus, A. Kulmala, E. Salminen, and T. D. Hämäläinen, "IP Integration Overhead Analysis in System-on-Chip Video Encoder," in *DDECS'07*, 2007.

[12] S. Zammattio, "SOPC Builder, a Novel Design Methodology for IP Integration," in *International Symposium on System-on-Chip*, Nov. 2005, pp. 37–37.