An Eclipse-based Framework for the IP-XACT-enabled Assembly of Mixed-Level IPs

Tao Xie, Gilles B. Defo, and Wolfgang Mueller
University of Paderborn /C-LAB, Fürstenallee 11, 33098 Paderborn, Germany
{tao, defo, wolfgang}@c-lab.de

Abstract. This article introduces basic functions of our IP-XACT based framework for automated design integration with mixed-level IPs and the key concepts behind the IP auto-assembly. The framework was generated from the SPIRIT standard XML data schema. Two examples, one with the CoreConnect™ SoC architecture and another one with an in-house library for FlexRay automotive bus simulation, are presented to illustrate the application.

1 Introduction

Meanwhile, we can find several data exchange standard which are based on the definition of a UML meta model and comparable descriptions like a XML data schema. Examples are AUTOSAR (http://www.autosar.org), DMTF (www.dmtf.org), and IP-XACT [2]. On the one hand, this facilitates the standardization process by the application of UML editors. Once the standard is defined as a UML metamodel XMI or a XML data schema can be easily generated from. On the other hand, this enables the generation of tools by the application of the Eclipse Modeling Framework (EMF) [5] which can easily generate model entries which then have to be further customized by specific code generator or advanced type checking, for instance.

In this context, we conducted an evaluation of IP-XACT with TLM and RTL IPs by building a standard-compatible Eclipse-based framework. With this framework we investigated the current capability and possible enhancement of IP-XACT for automated mixed-level SoC integration. Figures 1 gives an overview of our work. The IP-XACT standard as the input of the whole process is introduced briefly in Section 2. As such we applied the Eclipse Modeling Framework (EMF) [5] to generate the a basic IP-XACT data editor. The EMF enabled us (i) to study the application of IP-XACT and (ii) to investigate IP-XACT extensions. We further extended the editor to a SoC integrator with code generation capability under some extra rules which we identified for the TLM IP auto-assembly. Section 3 is devoted to the presentation of this framework. After that, two experiments, one with a typical SoC architecture and another from the automotive domain, are carried out and reported in Section 4. Here we use components at both RTL and TLM levels, from different parties to exercise better the framework. During this process specifically, the incorporation of several modeling environments, including the EMF, IP-XACT, and SystemC, are also evaluated. Conclusions are drawn in Section 5 where we identified several other
possible extensions for the IP-XACT standard, covering design aspects like power management.

![Figure 1: Overview of the work](image)

2 The IP-XACT standard

The major part of the IP-XACT standard [3] is a set of XML schemas as a meta-model, which defines the data format that is instantiated and delivered as XML files together with the IPs or their design integrations. This well-defined data format eventually enables the creation of vendor-independent tools for automated IP-integration.

Four central elements are defined, the bus definition, the abstraction definition, the component, and the design. A bus definition describes a bus, such as a typical SoC bus AHB or PLB, in terms of its most basic properties such as the supported number of masters and slaves. Further, an associated abstraction definition provides a complete description of the set of interfaces of the bus at a certain level of abstraction, e.g., TLM. All interfaces, such as the master and slave interface, are specified with their respective ports, at RTL or TLM.

A component packages an actual, potentially configurable, IP-core. It describes the bus interfaces exposed by the core, and the mapping between component ports and the respective ports from the corresponding abstraction definition. In addition, model parameters that enable the configuration of the IP core can be defined. Then an IP-XACT design describes an actual design by simply instantiating the components, configuring their parameters, and interconnecting them through bus interfaces, or individual ports directly.

3 Framework for IP-XACT-enabled Auto-Assembly

Our investigation of IP-XACT is based on its latest public release, the version 1.4. For this, we create a framework by applying the automatic framework generation process of the Eclipse Modeling Framework (EMF), which takes meta-models defined by an XML Schema as input and outputs an elementary modeling framework.
Following this scheme, we feed the EMF process with the complete XML schema of IP-XACT v1.4 and obtain a graphical editor for IP-XACT documents. Functionalities directly generated by EMF are creation, manipulation, and validation of any type of IP-XACT XML files. Since Eclipse is widely used for framework generation, this supports the incorporation of further extensions.

A simple screenshot of the editor is shown in Figure 2. We extend the basic editor by code generation capability. The code generator currently manages on SystemC IPs and their integration, at both RTL and TLM. The interconnections of modules are mainly done at the TLM level, though RTL ports are also dealt with. On hand with such extension efforts and the accompanying experiments, we were able to identify some lacks of IP-XACT for a full automation of design integration, especially at the TLM level, which is rarely addressed so far.

Figure 2: Screenshot of the IP-XACT entry

Figure 3 is an illustration of one such advanced modelling rule for IP-XACT that is necessary for automated TLM integration. It enables the automated binding of two TLM ports. We consider the TLM interfaces as a non-private inheritance tree, as depicted on the left hand side. The right side are two TLM ports, one implementing \texttt{IF\_6} and another expecting \texttt{IF\_2} for binding. For such a case, the following requirements are made:

- \textit{IP-XACT-provides} description for a TLM port includes all its inherited TLM interface class names and the \textit{IP-XACT-requires} description includes only the expected TLM interface for binding, and then

- An automation process for SoC integration decides the compatibility between the TLM port pair by determining \textit{whether provides-port includes the requires-port}.

In Figure 2, \texttt{port\_2} must describe all its inherited TLM interfaces -- \texttt{IF\_6, IF\_3, IF\_4, IF\_2}, and \texttt{IF\_1} in IP-XACT. Comparing them against \texttt{IF\_2}, the framework can judge the compatibility and know how to bind the two TLM ports safely, i.e. by typecasting at \texttt{IF\_2}. 
Following the original IP-XACT format, its associated semantics, and our extended TLM level rules, SystemC IPs and designs can be described and fed into the code generation flow, which is depicted by Figure 4.

The code generation engine takes an IP-XACT Design description as input and executes the following subtasks:

- Checking the involved IP-XACT descriptions, including the IP-XACT data validity, the original IP-XACT semantic rules (e.g. Bus/Abstraction Definition conformance), and our defined TLM IP-XACT modeling rules.
- Generating a SystemC design top file, consisting mainly of module instantiations, their parameter configurations, and inter-module port bindings.
- Generating a Makefile, by collecting IP meta information, like source files, library dependencies, etc, from their IP-XACT Component descriptions.

At the end, this composed Makefile is ready for a direct make simulation, which compiles all referred IPs and the SystemC top design, and launches without intermission the simulation.

4 Application Examples

We show two application examples of the framework. The first experiment applies the CoreConnect™ SoC architecture as an integration platform, using a comprehensive TLM modeling library for CoreConnect offered by IBM [4]. It
comprises a complete set of SystemC TLM IPs for the architecture, including the TLM Processor Local Bus (PLB), On-chip Peripheral Bus, bridges, PowerPC ISS, etc. At RTL level, we employed a RTL SRAM model from a second party with its adapted TLM/RTL transactor for the PLB bus. We then conducted two integration scenarios as shown in Figure 5. At first, four components, the TLM `plb_master_generic`, `plb_bus`, the RTL SRAM and its transactor, are instantiated and connected in an IP-XACT design. Then in a second scenario, a UART component is incrementally attached to the bus. For both scenarios, the address spaces are accordingly configured and when fed into the code generator, the two IP-XACT designs derive two seamless simulations of the top models.

The second case study is the assembly of simulation designs based on TLM FlexRay controllers. FlexRay [1] is a high data-rate, reliable automotive network communication protocol. As depicted in Figure 6, three TLM components of Electronic Control Units (ECU) communicate with each other via a FlexRay bus, each through a Flexray Communication Controller (CC). Abstract physical bus for Flexray exchanges data with the CCs through TLM sockets and TLM port-interface are used to model ECU-CC connections. We can configure the IP-XACT design by setting parameters like the number and IDs of Flexray CCs, generate accordingly the design models and start the simulations.

5 Conclusion

We implemented an experimental framework to validate the IP-XACT standard version 1.4 for automated IP assembly as IP-XACT [3] is a widely accepted standard.
for describing common electronic design IPs and their integrations. At version 1.4, which is the latest public release, its meta-model covers not only the RTL level but also the next promising Transaction Level Modelling (TLM) level. Though at RTL this is already a well-established approach, reuse of TLM IPs is still at its initial stage. The interoperability of TLM modules is emphasized with the actual TLM 2.0 [2] standard by defining strict modeling techniques to be followed by IP developers. The EMF based approach enabled us (i) to study the application of IP-XACT, (ii) to investigate IP-XACT TLM and other system level extensions, (iii) study the semantics for IP integration, and (iii) to study the dependency of the language extension to the code generation. The framework imposes additional TLM IP-XACT modeling rules on IP description and design integration, in order to obtain an advanced automation step. Our approach is different from a latest related work [6], which extends IP-XACT by extra schema definition to support a “Transaction Accurate” abstraction level. In contrast, we have focused on the automation with IP-XACT standard at TLM.

Two application examples show the capability of the framework, employing IPs from both RTL and TLM levels. They cover not only a typical SoC integration case but also a novel application of TLM modeling for the automobile bus and controller simulation. We currently investigate the extension of the framework to integrate software components with the generation of a SystemC/QEMU cosimulation. Other extension like the integration of hardware-in-the-Loop and the inclusion of IEEE UPF (Universal Power Format) information will follow. Due to our experience the main effort for such integration will go into the implementation of the code generation.

Acknowledgments. The work described herein is partly funded by the DFG through the SFB614 'Self-optimizing Concepts and Structures in Mechanical Engineering' and by BMBF through the SANITAS (01M3088) and the ITEA2-project VERDE (01S09012).

References