

SPIRIT IP-XACT Controlled ESL Design Tool Applied to a Network-on-Chip Platform

Emmanuel Vaumorin and Maxime Palus
Magillem Design Services
4 rue de la Pierre Levée
75011 Paris, France
{vaumorin, palus}@magillem.com

Fabien Clermidy and Jérôme Martin
CEA Leti – Minatec
17 rue des Martyrs
38054 Grenoble Cedex 9, France
{fabien.clermidy, jerome.martin}@cea.fr

ABSTRACT

Network-on-Chip is a very active field of research of the recent years. Compared to classical bus-based communication schemes, it implies innovative mechanisms as well as new ways of wrapping Intellectual Properties, giving more communication capabilities. To deal with the system design complexity, improved Electronic System Level design environments are needed. The purpose of this paper is to evaluate the benefits of an IP-XACT-based environment applied to Network-on-Chip design. We show the level of automation achieved in the design flow, discuss its efficiency for the design and verification steps, and propose improvements.

Keywords

ESL, Electronic System Level, NoC, Network-on-Chip, SPIRIT, IP-XACT, SoC design.

1. INTRODUCTION

Network-on-Chip (NoC) platforms are an alternative to the well-known bus architectures [2]. Offering a communication-centric approach of a design, they aim at overcoming the limitations of buses thanks to a better wire efficiency and a support for new communication-centric schemes [4]. Thanks to the NoC paradigm, both the design and application mapping are claimed to be simplified.

However, before this dream becomes true, efficient methods for NoC implementation have to be set-up. Indeed, NoC architectures address complex Systems on Chip (SoC), which possibly enclose dozens of IP cores. Evaluate the most appropriate NoC topology, plug the IP cores on it, simulate and validate the performances of the obtained design, and possibly change IP cores' relative positions on the NoC to improve efficiency, are typical challenges met by a NoC-based SoC designer. In order to handle this complexity and explore the potential design space at affordable effort and time, dedicated tools have to be used.

Numerous solutions are proposed in the literature. The Polaris framework [8] offers a complete development chain including tools for application traffic modeling, high-level design exploration, and backend-level projections and validations. Its design-space exploration plays on the NoC topology and on its Quality of Service (QoS). A similar tool suite has also been exhibited for the xpipes architecture [7], including NoC synthesis and technology projection. As regards the Ætheral architecture [3], its associated tools offer monitoring features for debug.

In this paper, we consider the FAUST2 platform, sequel of the FAUST one [5]. It proposes a data streaming communication model to guarantee the homogeneity of data transfer management for all IP cores. An adaptable Configuration and Communication controller (CC) ensures the interfacing between IP cores and the NoC. Unlike the NoC solutions presented above, where the representations of IPs are tool-specific, the chosen approach to handle the complexity of FAUST2 NoC design, and improve the time to validation, was to use a standard and unified representation of the whole system to ensure information consistency at every level of the design flow. The point was to secure the error-prone operation of rewriting the description of a single IP for different purposes, which often leads to mismatch between the different versions.

The IP-XACT standard for IP description [9] aims at providing SoC designers with such a unified model. It is an XML based open standard meant to target the needs of industry, defined by the SPIRIT consortium. This non-profit organization provides a unified set of specifications for documenting IPs using meta-data. These meta-data can then be used for configuring, integrating, and verifying IPs in advanced SoC design and interfacing tools using normalized APIs. They can be used to access design meta-data descriptions of complete systems.

To evaluate the benefits that IP-XACT could bring to FAUST2-based SoC design, an IP-XACT compliant toolset called Magillem has been chosen. Magillem supports advanced functionalities defined by the standard, like the ability to run code generators based on IP-XACT APIs, and facilities like a graphical design editor, tooling for IP import and packaging, design assembly and flow control.

The FAUST2 NoC platform is detailed in the next section. The main configuration parameters are extracted in order to point out the design complexity. Section 3 shows how IP-XACT can be used to set up and control a complete ESL flow according to a four-step strategy: library packaging, design assembly and verification, flow control, and advanced flow architecture. Section 4 then presents the work realized to adapt and customize the Magillem framework to the FAUST2 NoC platform. Finally, obtained results and limitations, as well as future possible extensions of the flow, are discussed.

2. FAUST2 PLATFORM

The FAUST2 Network-on-Chip architecture associates to each IP core a complete Communication and Configuration controller

(CC) (Figure 1). This section describes its main features. Flexibility of the proposed architecture is highlighted, and the ESL flow requirements are deduced.

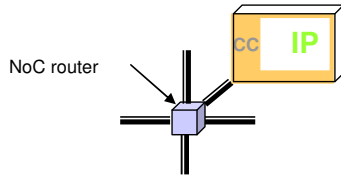


Figure 1. IP integration in the FAUST2 NoC.

2.1 CC Overview

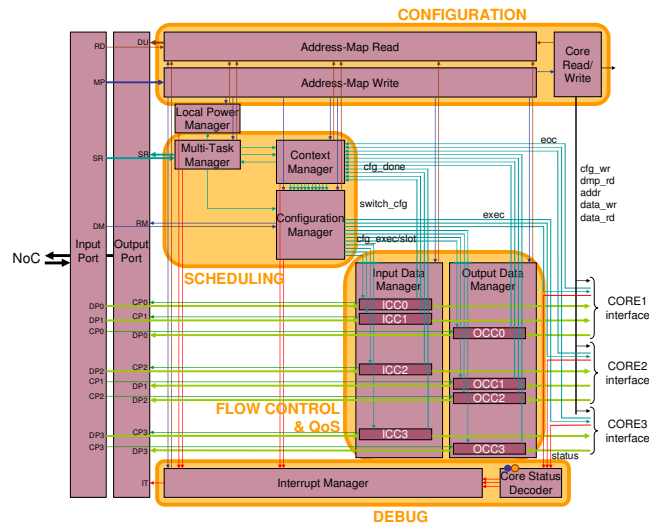


Figure 2. Communication Controller overview.

Figure 2 shows a typical example of a CC. Four parts can be distinguished, each made of several subcomponents with strong interactions between them:

- Communication management, including flow control, QoS, as well as communication scheduling features, allowing distributed communication management.
- CC core configuration management is able to handle not only static or off-line configuration, but also dynamic: a configuration may be loaded inside the IP core only when needed.
- Test & Debug features, provided by a test wrapper and by runtime traces and dump mechanisms, that allow precise control of an application's progress.

2.2 Core/CC Interface

To be connected to a CC, an IP core has to match the interface shown Figure 3. This interface is composed of a classical address/data configuration port, inputs and outputs for data flows, execution/status signals to start and control computation within the IP core, and some subsidiary signals, e.g. for test purposes. Up to four cores can be associated with a single CC, which obviously impacts the CC: it modifies the number of input and output flows, and also some internal functions. Finally, the wiring between CC's blocks is also impacted.

Moreover, and depending on the IP core (simple hardwired functions up to complex reconfigurable cores), some interface signals can be omitted and the width of some others can be modified. For example, BIST signals are necessary in case of memory blocks presence, whereas *size_released*'s width depends on the core management of its memory and can vary a lot from one core to another.

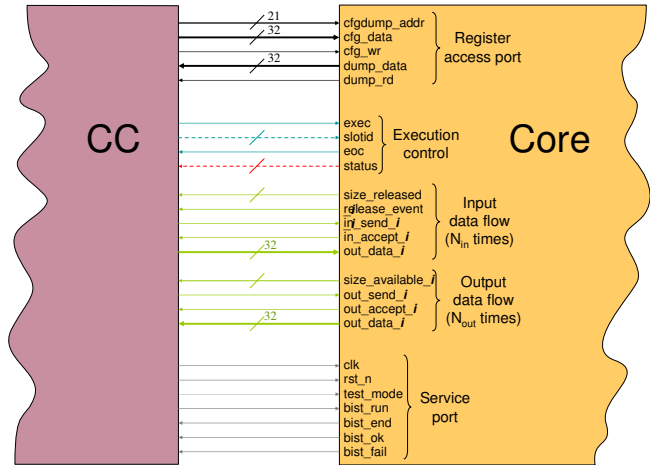


Figure 3. CC and core interface.

2.3 Communication & Flow Control Features

Apart from classical features of a network interface (e.g. message building, flow control and QoS), the CC provides an advanced integrated communication scheduler. The communications as well as their sequence, are interpreted and played by the CC, so that complex operations can be performed without the need of intermediate reconfigurations by an external controller, e.g. a CPU core.

Depending on the IP core, the CC can manage up to 4 input and 4 output flows, which modifies the number of blocks of the CC (e.g. numbers of OCCs, see Figure 2). The number of configurations, as well as the complexity of the scheduling, are strongly dependent on both the IP core and its use within the complete SoC: the same IP core may be associated with different CCs, depending on the functions it realizes in the application flow.

2.4 Reconfiguration Handling

The downsides of a flexible communication controller are (1) an IP core may have to be reconfigured during a communication sequence, in order to realize the global applicative sequence and (2) the number of required configurations, either for communications or for IP cores, may be very small or quite big. To solve the first point, a scheduler of IP core configurations, that supports the same sequences as for communications, is integrated in the CC. The second point raises the same problem of configurations storing for IP core as for the communications, in order to play a complete sequence. Depending on both the number of core registers to configure and the number of different configurations needed, the required memory might be huge, or in the contrary very small. The FAUST2 approach to solve this problem consists in a configuration cache mechanism, the CC and the IP core are able to store one or several configurations, and when a cache-miss occurs, i.e. a needed configuration is not

stored locally; the CC is able to automatically request it to a specialized IP core.

The cache size of the core and the corresponding control are therefore configurable. Core's multiple configurations are handled through a *slotid* signal (see Figure 3) which is an optional feature.

2.5 ESL Flow Requirements

As showed above, the particularity of the CC resides in its high level of flexibility: the number of cores and input/output flows, the communication and configuration complexity, and test capabilities are examples of features which can be set at design-time to ensure a perfect matching between the IP, the capabilities of its associated CC and application-level requirements. Such an approach avoids over-sizing of communication-dedicated components, saves power and improves performance. The counterpart is the necessity to have a highly capable and flexible design environment. High-level descriptions, such as SystemC/TLM¹ [6] must also be supported in order to accelerate the simulation of complex systems.

From a NoC generation point of view, the requirements of a design suit are: (1) to deal with optional signals and blocks, (2) to support different widths for a signal, (3) to be able to modify the parameters of each CC subcomponent, and in certain cases to generate different functions for a same block, (4) to connect the subblocks to obtain the correct CC, (5) to handle different representations of a same component and (6) to permit the final integration of the considered components in a complete design.

In other words, the tool suite has to be able to offer an efficient access to all the design parameters and functions, and to have a unified representation for all the models describing the blocks. The next section presents the IP-XACT standard, which is theoretically able to fulfill the discussed objectives. Section 4 relates the experience of an IP-XACT-based design flow for the FAUST2 NoC platform.

3. IP-XACT FOR ESL DESIGN FLOW

3.1 Overview

IP-XACT from the SPIRIT consortium is nowadays recognized by the electronics community as an apposite choice for managing properly and efficiently the new ESL design flows [1]. Nevertheless, the migration from a legacy design flow to another taking full benefits of IP-XACT requires some heavy and complex operations. Figure 4 presents the four steps which have to be completed. They are detailed in the following subsections.

3.2 IP Description

The goal of this first step is to package all the components of an IP library into XML files in accordance with the IP-XACT schema, which describes the syntax and semantic rules for the description of three kinds of elements: the bus definitions, the components and the designs (in which components are instantiated). Thus the purpose of the IP packaging is to fill in for each component the XML fields that describe its attributes: physical ports, interfaces, parameters, generics, register map, physical attributes, etc. An important part of the schema is

¹ TLM: Transaction Level Modelling.

dedicated to referencing the files related to the different views of a component: a view may be for instance a simulable model in a specific language (VHDL, Verilog, SystemC, etc) or documentation files (e.g. PDF, HTML, Framemaker). This work facilitates future reuse of existing components, because all of their features are easily accessible for its integration and configuration in a bigger system, as it will be explained in the next step.

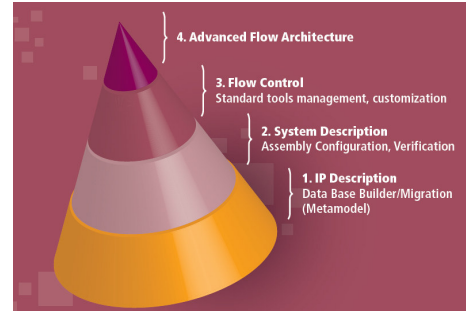


Figure 4. A 4-step methodology to build ESL flows.

3.3 System Description and Verification

After the packaging step, is it possible to import, configure and integrate components into the system, assemble the design, resolve connections issues, and automate design tasks, thus lightening the verification steps. Some example of the use of IP-XACT at this level are:

- Partial or full automation of design assembly and configuration, through TGI²-based generators that can instantiate, configure and connect components according to chosen design parameters (e.g. abstraction levels of components, type of architecture, etc.).
- Detection of communication protocols mismatch, thanks to the bus interface management, with possible insertion of the required adaptors/transactors.
- Generation by a TGI generator of the complete netlist defined by an IP-XACT design, e.g. in SystemC or VHDL.
- Automatic customization of compilation and simulation of designs. Indeed a component's description includes its entire related file path for each of its views (TLM, RTL, etc.), so a generator may build makefiles, apply potential component-specific compilation tags, and launch the compiler or simulator with the appropriate command line.

3.4 Flow Control

The third step of the methodology, depicted in the next figure, aims at linking the design activities around the centric IP-XACT database by means of a dedicated environment which provides access to the IP-XACT information. The Magillem tool provides an IP Packager, a Platform Assembly tool, as well as a Generator Studio to develop and debug additional TGI-based generators. These may be encapsulated within the IP-XACT representation of an IP and may for example simply launch the execution of a script, getting arguments values from the design description in

² TGI: Tight Generator Interface is the name of the API defined by SPIRIT for accessing data stored in an IP-XACT database.

IP-XACT, or be on the contrary a more complex engine, the role of which would be to modify the design itself (e.g. add connections, insert adapters, or configure components).

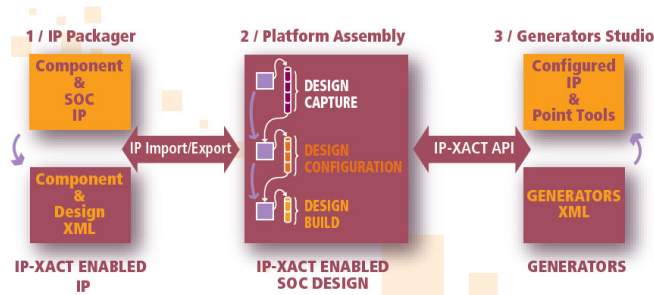


Figure 5. Principle diagram for an IP-XACT flow.

Checkers can also be developed and used to verify design rules at some point, before going further in the design flow. Besides, IP-XACT provides mechanisms to describe the sequences of chained generators and checkers.

3.5 Advanced Flow Architecture

This last step in the methodology has a high potential because it exploits all features described previously and allows the actual implementation of advanced ESL activities, such as architecture exploration or software application automated mapping on a hardware platform. These examples show the complexity that has to be managed by the three first steps: all components must be packaged and their configurability must be taken into account; the design assembly automation should be maximized, while any architecture choice should be handled. At last, the generator chains, as defined previously, can be configured and controlled by supervisor engines: for instance a validation sequence will configure and execute several times the generators dedicated to testbench configuration, compilation and simulation.

4. IP-XACT FLOW APPLIED TO FAUST2

4.1 Presentation of the ESL Design Flow

The analysis of the design flow used for the FAUST2 platform (IPs, tools, methodologies, documentation, etc) has led to the definition of five activities to be set up for the dedicated IP-XACT flow, presented in Figure 6 and detailed hereafter.

- Project management: description of the project's folder structure, path location of tools, project's parameters management.
- IP-XACT packaging of the library: extraction of IP data in folder structure and creation of metadata files.
- NoC assembly: generation of the units' interfaces, generation of the network, configuration of the routers.
- Compilation: setting of parameters, creation of compilation projects (makefiles) taking in account the context (TLM/RTL languages), compilers execution.
- Simulation & performance analysis: parameters interface, management of a simulation project, launch of simulations, extraction of results and back annotation in IP-XACT for analysis.

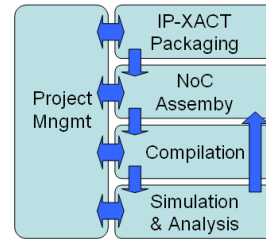


Figure 6. ESL design flow for FAUST2.

4.2 IP-XACT Packaging of the TLM and RTL Components

The packaging process starts with the definition of the communication protocols between modules. That means that groups of physical ports which belong to a same protocol are defined (IP-XACT *busDefinition* object). The direction of each port is specified for a target (slave) and for a source (master) use of the considered protocol. Some other information can also be stored, like the width of a port, default values, timing constraints, etc. These *busDefinitions* have been created manually for the FAUST2 platform using the Magillem integrated IP-XACT editor.

Regarding the packaging of the interfaces of RTL components to create their IP-XACT representation, it has been automatically done by Magillem with a parsing process able to extract the information from the VHDL model files. For the TLM components, this step has been done with the IP-XACT editor, which has also been used to update the representations with complementary information like register representation, IP-XACT generator inclusion, definition of specific parameters, etc.

After the packaging step, the IP-XACT components can be instantiated and connected in a graphical editor to create complete systems or hierarchical components.

4.3 CC Automated Generation

The FAUST2 CC, introduced in section 2, is made of 13 submodules with a high level of parameterization: number of cores interfaced by the considered CC, number of data inputs and outputs, configuration memory size, core status signal width, etc. These parameters allow the tuning of the CC to match the needs of the connected cores. In the rest of the paper, the term "specific" qualifies an element that has been configured according to the parameters chosen by the designer, as opposed to a "generic" element.

The whole CC generation process is handled by Magillem. An IP-XACT description of a generic CC, with the minimal interface, has been created, which encloses a generator able to create a specific CC. Generic IP-XACT components have also been created for all the submodules of the CC. They contain the interface, the memory map and a generator to create the corresponding specific CC submodule.

The generation of a specific CC is the result of the execution of a set of generators written in Java (relying on an extension of the IP-XACT TGI API) and Perl languages. The extended API adds Magillem specific functions like VHDL netlisting of an IP-XACT design and graphical manipulation of design representations

(instances and ports location and colors, specific logos for components), etc.

First of all, a generic CC is instantiated in a design and its embedded generator is called. This generator creates both the IP-XACT and VHDL descriptions of the specific CC, matching the chosen design parameters. As regards the IP-XACT model, information about the interface, the memory map and the VHDL model file set are captured in an IP-XACT component, whereas structural information (subcomponents, connections) are captured in an IP-XACT design.

Each submodule of the specific CC is then generated by instantiating the corresponding generic submodule in the CC design and running its embedded generator, which performs the following operations:

- Creation of the specific IP-XACT submodule.
- Call of a Perl generator which uses a generic template of VHDL code to creates the specific submodule's RTL model.
- Substitution of the generic IP-XACT component with the generated specific one.

Then the connections between submodules and to external ports are added to the CC design, and the global CC memory map and file set are created by collecting the information in all specific submodules, thereby completing the IP-XACT model of the specific CC. Figure 7 shows a fully-generated IP-XACT design as it appears at the end of this process.

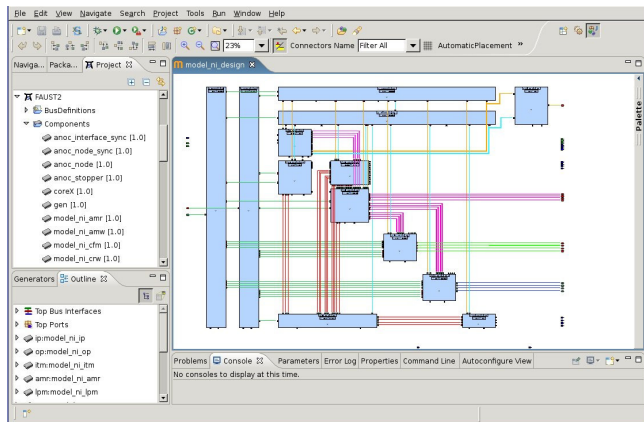


Figure 7. Graphical view of the IP-XACT design of a CC.

Finally the VHDL representation of the complete CC is automatically generated by the tool, which assembles the previously created specific VHDL components.

4.4 Design Assembly Automation

The CC generation is not the only automation provided by the use of IP-XACT tools and generators applied to the FAUST2 platform. Another TGI generator allows encapsulating and creating IP-XACT views of both SystemC/TLM and VHDL models of the top design of a complete FAUST2-based SoC. It uses a text configuration file that contains the wanted topology of the NoC interconnect and the name of IP cores that should be plugged on it. The SystemC/TLM model of the NoC is created at the same time by an external generator, along with a set of

configuration files used to program and test the described SoC. Besides, a VHDL netlister permits to get the corresponding RTL model of the whole SoC in a simple push-button manner.

The same top design generation mechanism is used to create TLM/SystemC simulation testbenches by adding or replacing some IP cores by debug-specific SystemC units. The user may also choose to simulate each considered IP core at TLM or RTL level, relying on external co-simulation tools.

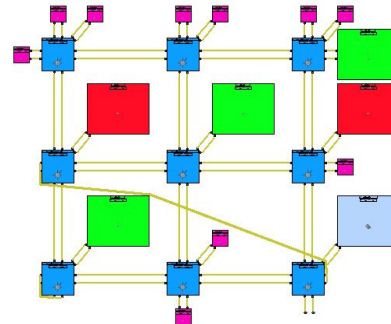


Figure 8. View of the IP-XACT design of a 3x3 NoC.

5.EVALUATION AND DISCUSSION

5.1 Benefits for the FAUST2 Design Flow

The IP-XACT ESL design flow presented in the previous section has been tested in order to create various testbenches of FAUST2-based systems. The main advantages pointed out during these tests are the ease of use, the unified model that references all information on the design components, and the reduced delay between the choice of the parameters and the complete assembly of the design.

Ease of use emphasizes the need of efficient IP-XACT tools such as the Magillem suite, which offers graphical representation and manipulation of IP-XACT models, hiding the verbosity of XML description files. From a designer point of view, it allows to browse through the design hierarchy to find and update any relevant information. Furthermore, generators can be run through the graphical user interface, and their results immediately seen in the tool. The automation possibilities, through configuration files, scripts and generators, also permits to hide the complexity of operations to the end user: when generating a CC for a core, he only has to enter the chosen parameters and get a complete CC after a few seconds.

The purpose of having a unified model that references all information about the components of a platform is to prevent redundancy of information between databases: it is quite common, for a SoC designer, to use different tools from various CAD vendors, each one dealing with specific information stored in different formats. In such cases it is difficult to ensure the consistency of the information, because when modifying some data used by one tool you possibly have to change the data used by other tools, this being a typically error-prone operation. IP-XACT offers the possibility to automatically reflect a change on all involved information.

Finally, the tests showed an important reduction in the design to validation cycle time. Indeed, when a new IP core has been developed in accordance to the FAUST2 core interface format

shown in Figure 3, it only takes a few minutes to import it and obtain its CC for a given set of parameters. Getting a complete testbench using this IP plugged on a NoC also is a matter of minutes. The designer may therefore concentrate on true value-adding tasks, like choosing architectural properties (NoC topology, memory size, multithreading support) and simulate the generated design to evaluate the performances. This allows a larger design space exploration than a manual parameterization of the testbench.

However the use of IP-XACT for the FAUST2 platform has also showed some limitations or weaker points which are presented in the next subsection.

5.2 Limitations

The most obvious drawback of adding IP-XACT to an ESL design flow is that it requires learning the IP-XACT format, integrating it into the previously used design database and packaging all used IPs. Even though this only has to be done once, the latter step may take a considerable amount of time, especially for complex systems. Indeed not all information may be taken into account by automated packagers, and most of the time some data, e.g. address mapping information, must be filled in manually.

Of course IP-XACT generators also have required several months to be developed and tuned to the specific needs of the FAUST2 platform, in order to achieve such a level of design automation.

On the other hand, frequently used commercial CAD tools do not currently support IP-XACT natively. This means that, to ensure a correct and automated transmission of design data to and from these tools, specific generators have to be developed and debugged.

5.3 Perspectives

The evaluation of IP-XACT potential benefits for the FAUST2 design flow will be pursued. The two main foreseen improvements deal with the link of the unified model with backend tools and with the embedded software development on the FAUST2 platform.

A link with backend tools would bring the opportunity to reflect in the unified model some characteristics calculated by the tools. For example, for a given core, power consumption to realise typical operations, and maximum computing performance, could be stored in the unified model, and used by high level TLM/SystemC models of a complete system to have realistic power and performance estimation for a complete application running on a SoC.

From the embedded software design point of view, the unified model already contains a lot of relevant information, especially regarding address mapping. A generator could easily solve the error-prone process of rewriting the address map according to the syntax of chosen programming language, as well as reflect automatically in software any change in the hardware address map.

6. CONCLUSION

In this paper, we showed how an IP-XACT-controlled ESL design flow may handle the design complexity of NoC-based SoCs. This

standard offers a unified representation of all relevant design information. In the typical FAUST2 case, it allows a quick integration of an IP core within the design, as well as an automated generation of complete systems.

However, the cost of switching from legacy to IP-XACT flows is not negligible, as it often requires manual operations to get a complete description of IPs. Moreover, native IP-XACT support by existing design tools is highly desirable, as for now generators have to be written to transfer relevant data to the CAD tools. Once these two points are solved, IP-XACT flexibility provides the designers with very valuable design flow customization and automation facilities.

7. REFERENCES

- [1] Bailey, B., Martin, G. and Piziali, A. 2007. ESL Design Verification. Morgan Kaufmann Publishers, 2007
- [2] Benini, L. and De Micheli, G. 2002. Networks on Chips: a New SoC Paradigm. IEEE Transactions on Computers 35, 1, (Jan. 2002), 70-78.
- [3] Ciordas, C., Hansson, A., Goossens, K., and Basten, T. 2006. A Monitoring-Aware Network-on-Chip Design Flow. In Proceedings of the 9th EUROMICRO conference on Digital System Design. DSD '2006.
- [4] Henkel, J., Wolf, W., and Chakradhar, S. 2004. On-chip networks: a scalable, communication-centric embedded system design paradigm. In Proceedings of the 17th International Conference on VLSI Design (June 21 - 24, 2004), 845 - 851. VLSID '04.
- [5] Lattard, D., et al. 2007. A Telecom Baseband Circuit based on an Asynchronous NoC. In IEEE International Solid-State Circuits Conference Dig. Tech. Papers (Feb. 11 - 15, 2007), 258 - 601. ISSCC '07.
- [6] Open SystemC Initiative (OSCI) homepage. <http://www.systemc.org>
- [7] Pullini, A. et al. 2007. NoC Design and Implementation in 65nm Technology. In Proceedings of the First International Symposium on Network-on-Chip. NOCS '2007.
- [8] Soteriou, V., Eislely, N., Wang, H., Li, B., and Peh, L. S. 2007. Polaris: A System-Level Roadmapping Toolchain for On-Chip Interconnection Networks. IEEE Transactions On Very Large Scale Integration (VLSI) Systems 15, 8 (Aug. 2007), 855 - 868.
- [9] SPIRIT Consortium homepage. <http://www.spiritconsortium.org>