

# Industrial IP Integration Flows based on IP-XACT™ Standards

Wido Kruijtzter<sup>1</sup>, Pieter van der Wolf<sup>1</sup>, Erwin de Kock<sup>1</sup>, Jan Stuyt<sup>1</sup>, Wolfgang Ecker<sup>2</sup>,  
Albrecht Mayer<sup>2</sup>, Serge Hustin<sup>3</sup>, Christophe Amerijckx<sup>3</sup>, Serge de Paoli<sup>4</sup>, Emmanuel Vaumorin<sup>5</sup>

<sup>1</sup>NXP Semiconductors, The Netherlands; <sup>2</sup>Infineon Technologies, Germany; <sup>3</sup>STMicroelectronics, Belgium

<sup>4</sup>STMicroelectronics, France; <sup>5</sup>Magillem Design Services, France

Wido.Kruijtzter@nxp.com

## Abstract

*Effective integration of advanced Systems-on-Chip (SoC) requires extensive reuse of IP modules as well as automation of the IP integration process, including verification. Key enablers for this are standards to describe and package IP modules. We focus on the IP-XACT standards and demonstrate how these standards are deployed in three industrial IP integration flows. Further, we report on two future extensions to IP-XACT that are currently being explored in the SPRINT project, i.e. IP-XACT based verification software generation and IP-XACT based configuration of debug environments. We conclude that IP-XACT is enabling powerful IP integration methodologies and that future extensions can further increase the effectiveness of IP-XACT standards.*

## 1. Introduction

The integration densities offered by nanoscale process technologies enable the development of advanced Systems-on-Chip (SoC). The complexity of developing SoCs is increasing continuously, but the productivity of hardware and software developers is not growing at a comparable pace. This is typically referred to as the Productivity Gap. As a consequence the costs of developing advanced SoCs are increasing at a staggering pace and time-to-market is negatively affected. The increasing complexity also has a negative impact on the quality of SoCs, and system verification and debug are becoming formidable tasks.

Key elements for addressing the SoC design complexity problem are IP reuse and extensive automation of design and verification activities. Enhanced interoperability and reusability of IP modules allows companies to share the costs and risks of developing IP modules, thereby avoiding duplication of development efforts. Extensive automation of the path from a SoC specification via integration of IP modules to an implementation improves time-to-market by reducing time-consuming and error-prone manual design and verification activities. The following aspects need to be addressed in order to enable reuse and automation:

1. Techniques for modeling IP modules. Specifically techniques for abstract modeling to enable fast simulations with high-level feedback, architecture exploration, early software development, and rapid construction of a reference model for verification.
2. Methodologies and tools to automate the integration of IP modules into SoCs, including support for correct instantiation and integration of IP. This should include generation of correct instances for configurable IP.

3. Methodologies, libraries and tools to automate the verification and debug of SoCs, including verification IP and support for automated generation of SoC verification suites. This should include reuse of verification suites and test benches over different abstraction levels.

Industry is already taking action to work on the aspects mentioned above. The OSCI consortium is progressing on the interoperability and reuse of high-level models via its SystemC/TLM working group. The SPIRIT consortium is contributing standards and technology to enable automation of design activities. It is developing the IP-XACT™ standard for IP description as well as for tools that raise automation levels. Further, 15 leading European Semiconductor companies, IP vendors, EDA companies and academic institutions are partnering in the European SPRINT project to develop new methodologies and standards for efficient reuse and exchange of IP. By building on existing initiatives such as OSCI SystemC/TLM and IP-XACT™ and feeding its results back into these initiatives, the SPRINT project is accelerating the evolution of these standards. To this end, representative players from the IP reuse value chain are closely collaborating, performing concrete design cases to generate requirements and evaluate extensions to the standards.

The paper is organized as follows: Section 2 gives a brief introduction to the IP-XACT standard. In Sections 3, 4, and 5, we present an overview of IP-XACT based industrial design and verification flows, as deployed by NXP Semiconductors, Infineon, and ST Microelectronics. In Sections 6 and 7 we report on two future extensions to IP-XACT that are currently being explored in the SPRINT project i.e. IP-XACT based verification software generation and IP-XACT based configuration of debug environments.

## 2. IP-XACT overview

The SPIRIT Consortium provides a unified set of high quality IP-XACT™ specifications for documenting IP using meta-data. This meta-data is used to configure, integrate, and verify IP in advanced SoC design environments. External tools, called generators, interface to such design environments using the LGI: Loose Generator Interface (database access through file exchange) and/or the TGI: Tight Generator Interface (database access through software API).

### 2.1 Typical IP-XACT based flow

The IP-XACT standard can be applied in various parts of a typical SoC design flow as depicted in Figure 1

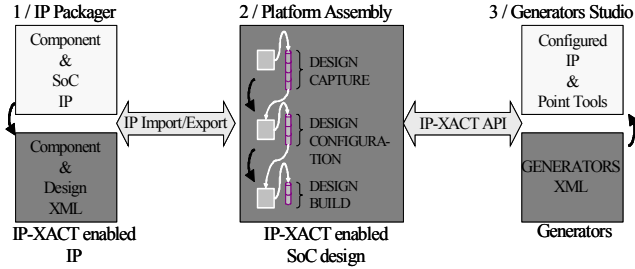


Figure 1: IP-XACT flow.

**IP Packaging:** The goal 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. It includes IP attributes such as physical ports, interfaces, parameters, generics, register map, etc. An important part of the schema is dedicated to referencing the various files related to the different views of a component: e.g. simulation model in a specific language or documentation.

**Platform Assembly:** After the packaging step, it is possible to import, configure and integrate components into the system, assemble the design, resolve connection issues, and automate design tasks. An examples of the use of IP-XACT at this level is the partial or full automation of design assembly and configuration, through TGI-based generators that can instantiate, configure and connect components according to chosen design parameters (e.g. abstraction levels of components).

**Flow Control:** The goal is to link the design activities around the centric IP-XACT database by means of a dedicated environment that provides access to the IP-XACT information. A typical tool suite provides an IP Packager, a Platform Assembly tool, as well as a Generator Studio to develop and debug additional TGI-based generators.

### 3. IP Integration flow of NXP

We present the IP-XACT-based IP integration flow for Electronic System Level (ESL) design and mixed RTL and ESL design in use in NXP. For the RTL IC design flow we refer to [4]. In [5] we describe SoC sub-system exchange.

#### 3.1 ESL Rationale and Practices

The purpose of ESL design is to improve the SoC design process in terms of decreasing design and verification effort, increase design quality, and reducing time-to-market. ESL models contain less information compared to RTL models. For this reason these models achieve higher simulation speeds and take less design effort. However, they are less accurate. Depending on the use case, we use different types of models as defined by OSCI (UT, LT, AT, and CA). NXP Semiconductors has adopted SystemC as language for ESL design already for several years. Our major use cases are architecture exploration, followed by software development, and validation and verification. ESL deployment in NXP so far has focused on virtual prototype deployment because integration and configuration of

SystemC IP is perceived to be difficult. The introduction of IP-XACT for ESL enables us to deploy SystemC IP throughout the company using NXPs' IP Yellow Pages and automate IP integration and configuration using our IP-XACT-based Nx-Builder tools in the same way as we do for RTL IP. This is essential for the deployment of a mixed RTL and ESL design flow.

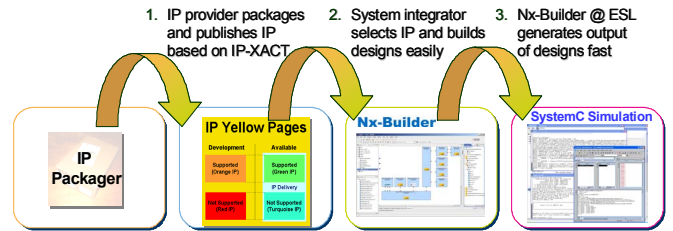


Figure 2: IP integration flow.

#### 3.2 Nx-Builder @ ESL Challenges

Our IP integration flow consists of three steps as depicted in Figure 2. In Step 1, IP providers package their IP in IP-XACT, i.e., they describe metadata of their IP in an XML document adhering to the IP-XACT XML schema. The IP and its metadata are published on the NXP-wide IP Yellow Pages. In Step 2, IP integrators select and configure IPs. They use an IP-XACT-based Design Environment to integrate the IPs using the metadata. In Step 3, tools generate outputs such as netlists, makefiles, and software. Specific challenges that are addressed by Nx-Builder@ESL that are supported by the IP-XACT 1.4 standard are packaging of ESL IPs, integration of ESL and RTL IP models, and generation of virtual prototypes and mixed level designs.

#### 3.3 IP Packaging

So far we have been packaging SystemC models of our IPs manually. We validated that we can package IP models that use our old proprietary modeling library as well as IP models that use the SCML modeling library. In the future we will adopt a commercial solution to package the SystemC models of our IPs automatically.

#### 3.4 IP Integration and Configuration

ESL models of IP at high abstraction levels are highly configurable because they are used for architecture exploration. We support configuration of such IP by IP-specific component generators. Such a generator takes the desired IP configuration as input and modifies the XML metadata description of that IP accordingly. We have chosen not to modify the XML description directly, but to make use of an API that provides access to the database of the IP-XACT-based Design Environment. Our experience is that this simplifies the writing of such component generators. In order to link ESL design to RTL design we make sure that the configuration of an ESL model of an IP can be transferred to a configuration of an RTL model of that IP. The Nx-Builder tool can extract both models in the desired configurations from the NXP IP Yellow Pages.

### 3.5 Virtual Prototype Generation

In order to generate output from Nx-Builder @ ESL we use a JAVA API that is generated from the TGI specification in the Web Services Description Language (WSDL). These generators can be linked against a Simple Object Access protocol (SOAP) implementation of the JAVA API in order to create TGI generators. We have developed generators that automate the creation and building of virtual prototypes. Amongst others they generate hierarchical netlists, makefiles for an NXP proprietary build flow, simulation scripts, memory map files, and verification software. For more information on verification software we refer to Section 6 of this paper.

## 4. IP Integration flow of Infineon

Infineon is a strong believer in the increasing demand of external IP integration and the leading role of the IP-XACT standards as enabler of automated IP-integration. For this reason, XML based code generation and IP integration is successfully applied in many Infineon projects since years. In the SPRINT project, Infineon brings all this experience to the project with the goal to accelerate the extension of the IP-XACT standard to serve Infineon's advanced needs.

### 4.1 Integration Verification

One focus of Infineon's work is the Quality Assurance (QA) of the generated code, especially the QA of the generated top-level netlists. This focus is motivated by the many and different optimizations performed in order to meet physical constraints of the silicon and performance and accuracy constraints of the models.

Two verification strategies are involved in Infineon's IP integration process: Direct and indirect verification. Direct integration verification verifies the correctness of the connectivity. Indirect integration verification verifies the correctness of actions executed via connections e.g. using read and write actions of registers in the IP devices. Since both RTL and TLM connections need to be verified, both formal verification (RTL only) and simulation is used.

### 4.2 Formal Integration Verification

Since the formal verification tools make themselves a complete search over all possible input patterns and states, the generation of the properties is quite straight forward.

1. Identity properties between two signals are specified for direct connectivity verification. These signals represent in most cases the valid address and the read/write/enable signals.

2. Register read/write properties assume a bus transaction at the bus master's side and checks versus the value of the register some cycles before or after. The assumption also includes interdiction of exceptions (e.g. reset) and hardware access to the registers during the write/read access.

### 4.3 Simulation based Integration Verification

Simulation based methods for integration verification follow the same approach as formal integration methods, however stimuli must be generated to drive simulation and

verification goals must be specified to be able to analyse the completeness of the verification runs. Stimulus generation primarily includes generation of transactions at master side, in a directed (read/write access to existing registers only), exhaustive (read/write to a wider address range), and constrained random way. Coverage analysis is done differently for the two verification strategies. Direct integration verification is measured in terms of toggle coverage of the involved signals. Indirect integration verification is measured in terms of register access

### 4.4 Tool flow

Figure 3 gives an overview on the tool flow. Starting from the IP-XACT description of the IPs and the top level, both the netlist in SystemC (including callbacks for the assertions) and the SystemC assertions are generated. Both are compiled together with the stimulus generator and executed as SystemC model. This simulation result produces coverage data (from execution of the assertions and from signal toggle analysis). This coverage data is finally compared with coverage goals, in order to validate the completeness of the verification.

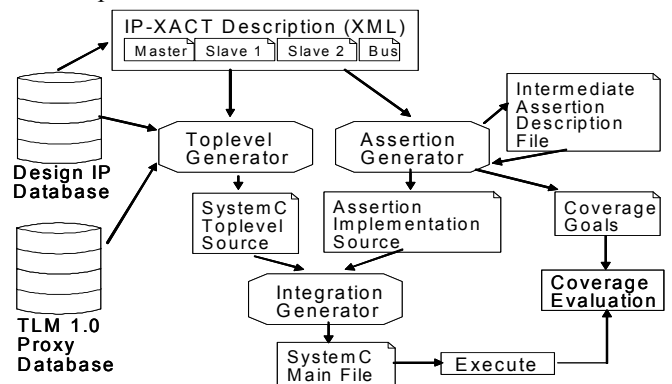


Figure 3: Infineon tool flow

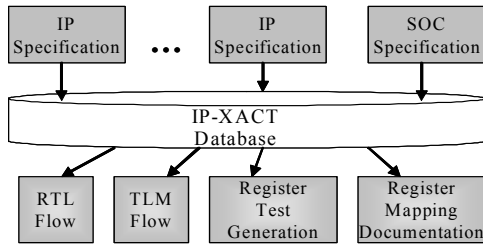
### 4.5 Implemented IP-XACT extensions

Key for verification is the link of the properties to the model and the generation of the assumptions required to set the multiplexers correctly. For this reason, we defined a set of requirements for IP views and usage of IP-XACT extension. These are:

1. A default view associated with bus definitions defines the read and write transactions on the bus. At the moment, we use different views for each assertion language involved, however we intend to have one abstract representation only. This view also ensures that exceptions are excluded on the bus for verification purposes.
2. Each IP must provide white box interfaces for the registers in the IP, and the wires connected to the registers.
3. Each register is extended with meta data describing the HW side, the actions performed by the HW side, and the protocols needed to perform that action
4. Each IP involved in the bus structure is extended with meta-data describing the translations made in the model.

## 5. IP Integration flow of STMicroelectronics

ST uses the flow shown in Figure 4 for the design and verification of its IPs, subsystems and chips.



**Figure 4 SPIRIT design and verification flow at ST**

This flow uses IP-XACT as a common format from which design groups can generate automatically e.g. the RTL netlist as described in [1], [2] the SystemC netlist used for the flow described in [3] the register test and header files that are used in design verification and software development; the register documentation for the product datasheets.

The IP-XACT database is made of IP-XACT compliant IPs and sub systems both from ST and from external suppliers. IPs built within ST are made IP-XACT compliant through a combination of conversion tools extracting the IP-XACT information from legacy formats e.g. the HDL files or the specifications. IPs from external suppliers are either provided in IP-XACT format or made IP-XACT compliant within ST.

Whatever the source of the IP it needs to pass a QA (Quality Assurance) step before entering the IP-XACT database. This QA step is fundamental to the flow as it guarantees the time won through design automation will not be lost debugging IP-XACT IP representations. It is also very illustrative of the type of flow automation we have achieved with IP-XACT as it uses many of the tools we are using around IP-XACT.

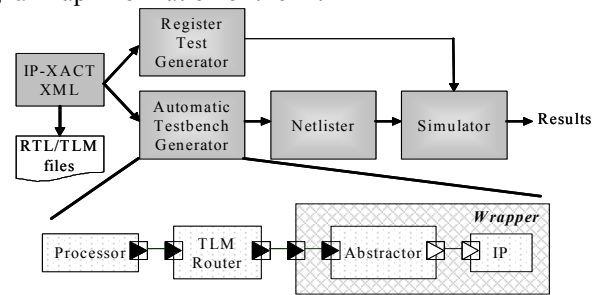
### 5.1 IP Quality Check Flow

The purpose of the IP QA flow is to verify that the IP-XACT description of an IP is accurate e.g. registers described in IP-XACT are indeed implemented in the RTL of the IP, or the bus interface signals can indeed be driven by the protocol they claim to support. A secondary purpose of the IP-QA flow is to detect unexpected incompatibilities with the IP-XACT generators used later in the flow by already using these tools in the QA flow. The outline of the QA flow is described in Figure 5.

Starting from the IP-XACT description of the IP, an IP-XACT testbench is generated, instantiating the IP and connecting it to a high-level processor model, written in SystemC, on which the test software is executed.

If the IP under QA is modeled at the SystemC TLM level, it is instantiated as such inside the testbench and connected to the processor via a high level TLM bus model. If the IP under QA is not modeled at the TLM level, it is first wrapped in a co-simulation wrapper made of abstractors bringing the IP RTL bus interfaces to TLM level. The abstractors are instantiated and connected

automatically based on the IP-XACT bus interface and signal map information of the IP.



**Figure 5: IP Quality Check Flow**

The SystemC TLM testbench is automatically generated by netlisting the IP-XACT test bench with a TLM netlister. Another generator is also automatically called to generate the register test from the memory map information inside the IP-XACT file. The current IP-XACT schema doesn't contain any information regarding side-effects. In order to have reliable results, we have our own IP-XACT vendor extensions to describe these side-effects. These vendor extensions are used by the register test generator to propagate the information inside the register test files.

Finally, the netlist and the register test are combined to run a simulation. A successful simulation demonstrates firstly that the file set section of the IP-XACT file is correct since we couldn't compile if it was wrong. Secondly that the communication between the processor and the IP is working properly (it checks that the signal and interface description is correct inside the IP-XACT file) and finally that the register information and our vendor extensions are aligned with the content of the IP.

## 6. IP-XACT based vSW code generation

For IP-based (sub-)system designs it is a tedious job to find low level interconnection bugs through functional simulation on application level. For that reason NXP first applies structural system verification in order to find design errors that are not related to an application, but only to the platform standard that is being used in the design. Examples are bugs in address decoding, DMA channels, interrupts, clock programming, and so on. Such methodology requires that IPs are supplied with their own verification software (vSW) in plain ANSI-C. This software can be used by a SoC integrator to verify that the IP was successfully integrated in the system design. It checks the connections of the IP to surrounding infrastructure IP.

The interfaces of any IP can be assigned to an interface class. Each of these (about 10) classes can be represented by a software API, which enables the IP to control or query the connected infrastructure IP. Most of the IP verification software can be generated automatically from the information in the IP-XACT component file of that IP, but what is still missing are tags on an IP interface that tell to which interface class it belongs.

## 6.1 Context labels Extensions

Context labels are defined for components and for their interfaces (i.e. individual ports and/or buses). For brevity we only show the interface labels. These are tags showing the intended use (design intent) of a bus interface or port. A system designer may have a different use (user intent) for an IP interface (e.g. a GPIO port normally used to drive an off-chip interface, but now used to control internal system signals); this should then be annotated to the IP instance in the system design.

When the hierarchy of an IP-XACT design is traversed, the constituting components are inspected for context labels. These context labels are attached to the components themselves, to bus interfaces or to individual ports. They can be overruled by similar context labels in a design file where they are attached to component instances, interconnections and ad-hoc interconnections, respectively. These context labels are formally defined as a SPRINT proposal to enhance the SPIRIT IP-XACT 1.4 standard to contain behavioral properties of an IP component or interface. Examples of context labels are `<isNetworkComponent>` for bridges, busses etc.; `<isInfrastructureComponent>` for Clk and Reset generators, Interrupt Controllers etc.

## 6.2 Automated integration verification generation

In the NXP flow the HW platform is built using RDL (Register Description Language) and HDL resources. To generate the integration verification setup a CSV (Comma Separated Value) file (excel sheet) is used that describes the system connectivity data. This CSV file is input to a script that generates the chip context required to manage the integration verification.

Magillem Design Services and NXP have worked on a methodology (Figure 6) that adapts the IP-XACT standard to automate the creation of this integration verification setup using vSW. The HW components are packaged in IP-XACT, allowing any compliant tool environment to build the platform and exploit the database by TGI generators. The intermediate CSV format is generated from the description of a design in IP-XACT.

Each component of the library is available through its IP-XACT description, which is referring the HDL/RDL files and the corresponding vSW view in C. The HW platform construction and elaboration for simulation is applying a common IP-XACT based setup and a netlister. In parallel, the generation of the chip context is done from the same IP-XACT design description. The simulation can be launched in a press button manner, as all the elements of the flow are packaged in SPIRIT generators linked in the generator chain.

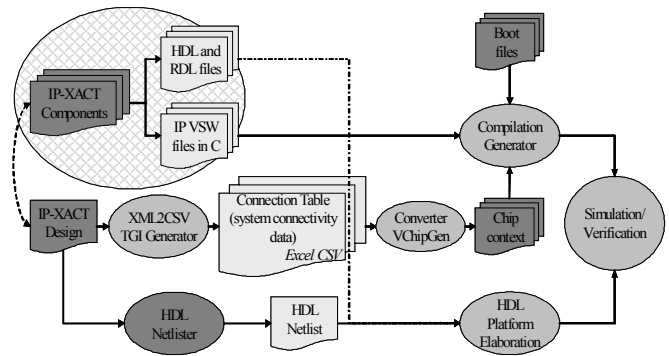


Figure 6 vSW Methodology

The validation of the presented flow and applied tools has been done on a real ARM9 based platform from NXP. The bus definitions, components and the design are packaged in IP-XACT using the Magillem environment. We have used the vendor-extension mechanism to extend the existing XML files with the context labels. The XML2CSV generator has been linked with the design description such that it can be launched through Magillem.

## 7. IP-XACT based debug environment config

The typical use-case described in this section is a debugger that is attached to a processor running firmware accessing peripheral IPs. Two aspects are studied in the SPRINT project. Firstly handling resources outside the processor core: This part is focusing on the added value from the debugger *user* point of view. The goal is to provide an enhanced view of the registers of peripheral IPs that are visible from the processor addressing space (Section 7.1).

Secondly handling resources inside the processor core: This part is focusing on the added value from the debugger *targeting* point of view. The goal is to reduce information duplication and to reduce the debugger targeting effort (Section 7.2).

### 7.1 Memory Mapped Resource Viewer

Peripheral IP registers are mapped in the processor memory space. The number of memory mapped registers may be very large. Therefore, having an enhanced view of these registers (e.g. both addresses and detailed register field descriptions) similarly as described in the specifications is helpful for the firmware developer. In ST, we have implemented a Memory Mapped Resource Viewer (MMRV) as an Eclipse plug-in, interfaced to the debugger perspective, and synchronized to the debugger (see Figure 7) in which registers are displayed in a hierarchical way: IP/Register/Field according to their IP-XACT description. The MMRV has been submitted to the Eclipse consortium. In ST the MMRV is now entering in its deployment phase; it is packaged within ST Software Development Toolsets.

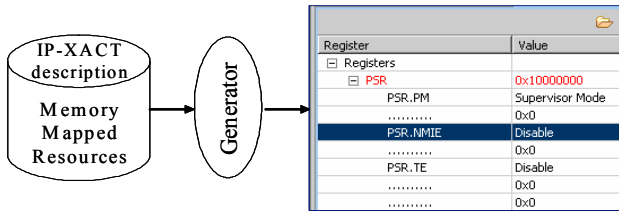


Figure 7: Memory Mapped Register viewer

## 7.2 Debugger retargeting

Retargeting a debugger is commonly performed by writing dedicated code to support the new processor core. This includes a “Resource Manager” describing registers and memories; an “Architecture Manager” describing the general architecture of the core; and a “Type Manager” describing the supported data types. A part of the required information for the targeting task comes from the architecture description. Using a standard way to retrieve this information allows retargeting a debugger with less effort. Figure 8 illustrates the objective of this task: Left side is a classical scheme of the target-dependent part of a debugger diagram; Right side explores how IP-XACT can be used to put the target description outside of the debugger core diagram.

IP-XACT allows describing registers and memories (i.e. address spaces). It supports the following register attributes: name, size, address offset, and fields, and address space attributes: name, address range, and width.

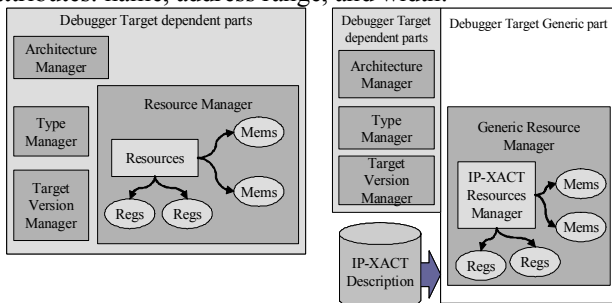


Figure 8: Debugger retargeting

The debugger manages the correspondence between the software and the hardware. Therefore it needs additional information. The following ones have been identified:

1. Defining register and memory identifiers to indicate correspondence between software and hardware.
2. Defining a display name for resources.
3. Defining pseudo registers, which do not exist in the hardware, but that do make sense in software.
4. Additional attributes to memories, defining whether they are data or program memories.
5. Additional attributes to registers, defining whether they are Program Counter, Stack Pointer, Frame Pointer; if they are hidden register (not visible to the user);
6. Defining the type and the format of the data being contained in the registers (integer or float). In case of a float format (e.g. IEEE 754) the different parameters of the format need to be described.

7. Describing the memory representation of data contained in registers. This is useful in case of a register spill (the register values are stored in the stack). In this case, displaying a variable when the program reaches a break point is not performed by reading the register, but by reading its value in the stack.

8. Specifying if a register value dynamically depends on the value of another register (e.g. a Status Register “mode” field that indicates if the core is running in 16 or 24 bits mode, meaning that the data contained in General Purpose registers are 16 bits or 24 bits).

These extensions are currently experimented in ST, using `<spirit:vendorExtension>`.

## 7.3 Discussion

Using IP-XACT description for configuring memory mapped resource handling from a debugger offers speed-up in the integration of an external IP. The debugger targeting cost for displaying the memory mapped registers of the IP in a processor address space is null (only need to load the IP-XACT description). From a user point of view, it provides both easy to read description of memory mapped resource fields, as well as a calculation of register addresses and a generation of the debugger commands. This reduces risks of human errors and increases the productivity of the firmware developer.

SPRINT proposes IP-XACT extensions for interpreting the data contained in the registers from a software point of view. The objective is to rely on a single description of the target which also allows automating the debugger targeting tasks.

## 8. Summary

Existing industrial IP integration flows already benefit from applying the IP-XACT standard. These benefits are in various areas such as: Automated SystemC IP Packaging, Integration and Configuration and Virtual Prototype generation (NXP), IP Integration Verification, both formal and simulation based (Infineon) and IP Quality Assurance (ST). In the future more elements of the SoC design flow such as verification SW generation, integration verification and debug environment configuration can be automated using next versions of IP-XACT. Several of the required extensions are currently prototyped in the SPRINT project.

## 9. References

- [1] O. Florent et al., “Spirit-based IP Assembly and SDC Promotion for a 65-nm System-on-Chip using coreAssembler”, in *Proceedings of SNUG Europe 2006*.
- [2] O. Florent and F. Remond, “65nm SOC design based on an emerging standard: Spirit”, presented at IP-SOC 2005.
- [3] F. Ghenassia, Ed., *Transaction-Level Modeling with SystemC: TLM Concepts and Applications for Embedded Systems*, Springer, 2005.
- [4] Geoff Mole et al, “Philips Semiconductors Next Generation Architectural IP ReUse Developments for SoC Integration”, IP-Soc 2004.
- [5] Strik, Marino et al, ” Subsystem Exchange in a Concurrent Design Process Environment”, Proceedings DATE 2008.