Exploring the Use of IP-XACT in a TLM Environment

Eirik Prestegårdshus

Master of Science in Electronics
Submission date: June 2017
Supervisor: Kjetil Svarstad, IES

Norwegian University of Science and Technology
Department of Electronic Systems

Problem Description

Title: Exploring the Use of IP-XACT in a TLM Environment

Date: 18.06.2017

Student: Eirik Prestegårdshus

Academic Supervisor: Kjetil Svarstad

Corporate Supervisors: Conrad Foik, Junaid Elahi

The IP-XACT standard presents a methodology to store metadata of electronic components. There are increasing efforts to automate design flows, which leads to metadata methodologies becoming increasingly valuable to the industry. IP-XACT's analysis consist of both the methodologies and the format to store the metadata.

The goal of this thesis is to explore the viability of using IP-XACT in the design flow, and additionally its relevance to a transaction-level modelling (TLM) environment. Research includes study, experimentation and analysis of IP-XACT.

The problem is provided by Nordic Semiconductor ASA.

Preface

This Master thesis is the final work of my Master's Degree in electronics at Norwegian University of Science and Technology (NTNU). The work is carried out at the Department of Electronic Systems. The project has great relevancy to the education, especially to the last few years sporting a more digital approach. The six months researching the thesis has been a very interesting journey, which has helped to grasp an understanding of one of the back-end systems in the industry. The project is proposed by Nordic Semiconductor ASA, which also provides all equipment and a workplace.

First and foremost, I would like to thank Conrad Foik for outstanding support and feedback to all aspects of the thesis. Without his advice, the study, discussions nor thesis would be nearly as interesting as it have become. Furthermore I would like to thank J. Elahi and L. H. Olsen for an insight to methodologies at Nordic, together with all other colleagues at Nordic that made the months of study and writing much easier to get through. I would also like to thank Vincent Thibaut for taking the time to answer many of my questions regarding the standard.

At last, I would like to express my true gratitude to my academic tutor Kjetil Svarstad who, albeit a very tight schedule, provided good feedback, rewarding meetings and generally kept me on track.

Sincerely,

Eirik Prestegårdshus, June 11th, 2017

Abstract

As both semiconductor technology and industry evolves, the need for robust methodologies and efficient design flows becomes more and more important. This thesis investigates IP-XACT, which is a standard that describes metadata of electronic systems aimed at computer-aided designs. The thesis emphasises methodologies of IP-XACT, as well as the overall structure and execution of said methodologies.

The thesis carries out an experiment where IP-XACT is used with transactional-level modelling (TLM). The experiment emphasises methodologies, implementation and structure of the standard. IP-XACT fully accomplishes to describe a component's metadata, and additionally serve as very valuable to the entire design flow and as a back-end metadata organisation. Study and experimentation show the methodologies and core metadata description to be strong concepts, but criticise lacking vendor neutrality and an unintuitive mixed-model structure. To solve the discovered weaknesses, the thesis proposes an alternative model that restructure elements that describe information specific to models (like TLM).

IP-XACT is ready to be used in the design flow, but is still a subject of development, thus feedback can prove very valuable for future versions. IP-XACT seemingly has a positive future to come, especially with the industry being more reliant on computer-aided design flows to achieve higher levels of efficiency.

Sammendrag

En voksende halvleder teknologi med tilhørende industri gjør at robuste metoder og effektive designsykler blir viktigere og viktigere. Denne avhandlingen undersøker IP-XACT, som er en standard som beskriver metadata for elektroniske systemer rettet mot datastyrte design. Avhandlingen vektlegger metodikken til IP-XACT, samt generell struktur og utførelse av nevnt metodikk.

Avhandlingen utfører et eksperiment hvor IP-XACT brukes sammen med en model på transaksjonsnivå (TLM). Forsøket vektlegger metodikk, implementering og struktur av standarden. Observasjoner viser at IP-XACT beskriver komponenters og systemers metadata på en bra måte, og fungerer bra som en singel kilde for metadata i designflyten. Studier og eksperimentet viser metodologien, ideene og kjernestrukturen som sterke konsepter, men kritiserer manglende leverandør-nøytralitiet og en lite intiutiv struktur for å beskrive forskjellige implementasjonsmodeller. For å løse de oppdagede svakhetene, foreslår avhandlingen en alternativ modell som omstrukturerer elementene som beskriver spesifikke implementasjonsmodeller (som TLM).

IP-XACT er klar til å brukes i designflyten, men blir fortsatt utviklet og forbedret, derfor kan tilbakemeldinger bringe store verdier for fremtidige versjoner. IP-XACT har tilsynelatende en positiv fremtid fremfor seg, særlig ettersom industrien blir mer avhengig av en datastyrt designflyt for å oppnå høyere effektivitet.

Table of Contents

Abbreviations

- API – Application Programming Interface
- CAD – Computer-Aided Design
- EDA – Electronic design automation
- GNU – GNU's Not Unix
- GPL – GNU Public License
- HW – Hardware
- IP – Intellectual Property
- LGPL – Lesser GNU Public License
- OSCI – Open SystemC Initiative
- OVM – Open Verification Methodology
- RNG – Random Number Generator
- RTL – Register transfer level
- SAM – System Architecture Model
- SOC – System-On-a-Chip
- SW – Software
- TLM – Transaction Level Modelling
- UVM – Universal Verification Methodology
- VHDL – Very high speed Hardware Descriptive Language
- VLNV – Vendor Library Name Version
- XML – eXstensible Markup Language

1 Introduction

More than fifty years ago Moore's Law predicted that the number of transistors per area grows exponentially every two years. The law still holds true today, and according to the industry, it will continue to be valid for the next couple of decades [1]. From a digital perspective, this means that more complexity will be packed into each chip. There is fierce competition in the semiconductor market, which follow this exponential trend. As complexity rises, design methodologies and design flow become more and more important. Time-to-market is an important factor as the technology evolves, and staying behind in design methodologies may lead to a company's downfall [2]. To tackle the growing complexity while keeping a competitive edge requires an efficient and productive design cycle. In terms of design, a high level of reuse and an efficient design cycle is required to keep up with aggressive schedules. IP-XACT proposes a solution to the problem, which emphasizes reuse and design integration, as well as a unified design methodology. The high complexity also pushes the need for the designer to take correct design decisions early in the design cycle, which is where IP-XACT is the most prevalent.

This thesis investigates IP-XACT, and its impact on the design flow methodologies. IP-XACT is a standard that describes component metadata conveniently for a computer-aided design (CAD) flow [3]. IP-XACT emphasizes reuse of existing components for a more unified and efficient design cycle. This thesis experiments with IP-XACT implemented with a Transaction-Level Modelling (TLM) design. TLM has the value of a higher abstraction that enables quicker and easier experimentation compared to RTL [4]. The experiment shows IP-XACT generally interpreted in a design flow and how it contributes to the TLM design flow specifically. TLM is likely to be less known for the front-end designer than RTL, but TLM's importance grows proportional to the complexity of designs and future predictions depict TLM to be a necessity for large-scale designs [5]. IP-XACT's level of abstraction is particularly interesting when applied to system-on-a-chip (SOC), aka large-scale designs, which makes the two a good match for research.

Important areas of research comprise of how IP-XACT can contribute to the design and increase design productivity. IP-Reuse and IP integration are two significant aspects of IP-XACT, it is important to emphasize how the standard implements those two. IP-XACT released in 2004, but is still a subject of further development [6], so there are multiple changes and additions coming in the future. The recent changes targets support for more complex system representation and conveniently adds more support for TLM. Chip complexity is only increasing in the future, thus the thesis also aim to lightly discuss future projections for IP-XACT. The contributions of this thesis are research and analysis of IP-XACT, an IP-XACT/TLM experiment, and discussion regarding IP-XACT in general, its application to the world and its internal structure.

1.1 Scope of the thesis

The scope of the thesis is IP-XACT within a TLM environment. Both IP-XACT and TLM are tightly coupled with RTL. However, this thesis will only discuss RTL lightly and experiment solely with IP-XACT and TLM. The reason behind that is that the representation of a component described by its metadata are almost the same whether the model is TLM or RTL. TLM provides the benefit of being faster to implement and simulate, also in a simpler and more accessible way than the RTL counterpart. The experiment does not aim to perform complex in-depth TLM experimentation, but discusses aspects where IP-XACT can provide benefit for a more in-depth TLM implementation along its use in the current setup.

The thesis emphasises the general design flow and methodology of IP-XACT. The design flow presented in this thesis is a simplified version of one in the industry, but still resembles many of the same cases and problems present to the industry. The thesis often discuss from a designer's point of view, meaning a designer that presumably works in a stage of the design cycle at a company. The thesis often use the wording *complexity* as a measure. In theory, complexity is the amount of functionality contained within a defined design space, but functionality can be interpreted both user-friendly and hard to understand; the impression of *complexity* changes accordingly.

IP-XACT is not coupled to any one language nor vendor, yet IP-XACT is only used in conjunction with a tool. The thesis aim to discuss the standard itself and not the tools that use it. The investigation of IP-XACT properties derive entirely from IEEE 1865-2009/2014 standards, and will to a lesser degree emphasise what may be added through *vendorExtension*. The core of the thesis' scope is to present IP-XACT's strengths and weaknesses, and more specifically in a TLM setting. To aid the discussion and demonstration of the standard; the thesis make use of an experiment.

1.2 Report Organisation

The report is divided into Theory, Experiment, Experiences, Discussion and Conclusion. The Theory provides relevant background information for XML, IP-XACT, TLM and the processor used in the experiment. The Experiment presents methodologies of design flow and setup of the experiment. Experiences describes observations from the structure and methodologies and observations of the experiment. Discussion is the main body of the thesis. The Discussion debates the methodologies and interpretation of IP-XACT both as ideas and execution, and from multiple angles. Lastly, Conclusion brings all of the results and discussions together to form a verdict to IP-XACT's current and future situation.

2 Theory

This chapter gives essential background information regarding the systems and methodologies used by the thesis. It introduces IP-Reuse, theory of the used programming languages, methodologies in TLM and the environment of the experiment.

IP-Reuse, as the name suggests, is to reuse a part of the design previously created by others. Others may refer to people within a design group, within a company or from an external provider. When talking about IP-Reuse there are two terms one should know, the types of Intellectual Property (IP) [7]. An IP can be delivered/described as either soft-IP or hard-IP. Soft-IP is synthesisable HDL [8], which means the internals can be viewed, debugged and potentially edited. According to the providers' licenses, one might not be allowed to edit the delivered code. The other deliverable is hard-IP, an IP described as technology specific layout that cannot be viewed nor altered. IP-Reuse most commonly refers to soft-IP, and this thesis further assumes IP-Reuse linked to soft-IPs. Furthermore, since the focus of the thesis is TLM, the soft-IP in the experiment describes in compileable SystemC [9], not synthesisable HDL.

2.1 Extensible Markup Language

Extensible Markup Language (XML) [10] defines a markup language that describes rules for encoding structured data. The specification of XML is governed by World Wide Web Consortium (W3C) [11], and serve as a worldwide standard. XML's goal is to be simple and to serve many purposes, as well as being easily readable by both humans and machines. XML has been widely adopted by the internet, and numerous XML schemas leverage data transfers across the web [12].

XML is like programming languages in the way that it has syntax that must be followed. However, XML and programming languages differ in the way they are used; XML is not executable. XML is merely a format to store information. The language is a textual language where the markups are tags. A tag can store information and have child tags, which frequently are structured hierarchically as trees. This thesis often use the wording element when describing a tag with its underlying tags. The format that defines allowed tags, and how they are structured are called a schema. IP-XACT is one of many standardised XML schemas.

2.2 IP-XACT

IP-XACT is an XML Schema that describes the structure of metadata in and about electronic components. IP-XACT is an open industry standard that goes by the code *IEEE 1685, Standard for IP-XACT, Standard Structure for Packaging, Integrating and Re-Using IP Within Tool-Flows* [13]. SPIRIT Consortium initially created IP-XACT in 2003, and as of 2009 SPIRIT merged with Accellera [14]. In June 2010 IP-XACT became an IEEE [15] standard, IEEE-1685 [13]. IP-XACT is currently being worked on by Accellera's working group, which has members across multiple big companies such as ARM [16], Magillem [17] and Xilinx [18]. The working group has lately developed a set of extensions for IP-XACT to describe analog/mixed-signal/digital designs, area estimates and support for power descriptions [6].

IP-XACT is able to describe the entire design, and splits design into eight different top-level elements. The following list describes each of the elements, be aware that additional top-level descriptions might be added in the future. The list below describes all the elements with a short description.

- A ***bus*** *definition* description defines the type attributes of a bus.
- An ***abstraction*** *definition* description defines the representation attributes of a bus.
- A ***component*** description defines an IP or interconnect structure.
- A ***design*** description defines the configuration of and interconnection between components.
- An ***abstractor*** description defines an adaptor between interfaces of two different abstractions.
- A ***generator*** *chain* description defines the grouping and ordering of generators.
- A ***design*** *configuration* description defines additional configuration information for a generator-chain or design description.
- A ***catalog*** description provides a mapping between IP-XACT VLNVs[1] (see 1.3.3) and the physical location of the IP-XACT file defining the IP-XACT object with the given VLNV.

IEEE Std 1685™-2014 [13]

XML allow IP-XACT to store data in a tree like structure. Figure 2-a describes the *busDefinition* element with all the possible sub-elements; elements with dotted boxes are optional. A set of checkers can run through each element to validate correctness of the contained data.

[1] Vendor Library Name Version (VLNV) is an identifier to uniquely reference an electronic component. They are often described as strings with colon as separators, an example of such VLNV may be *amba.com:AMBA3:AHBLite:r1p0_6*.

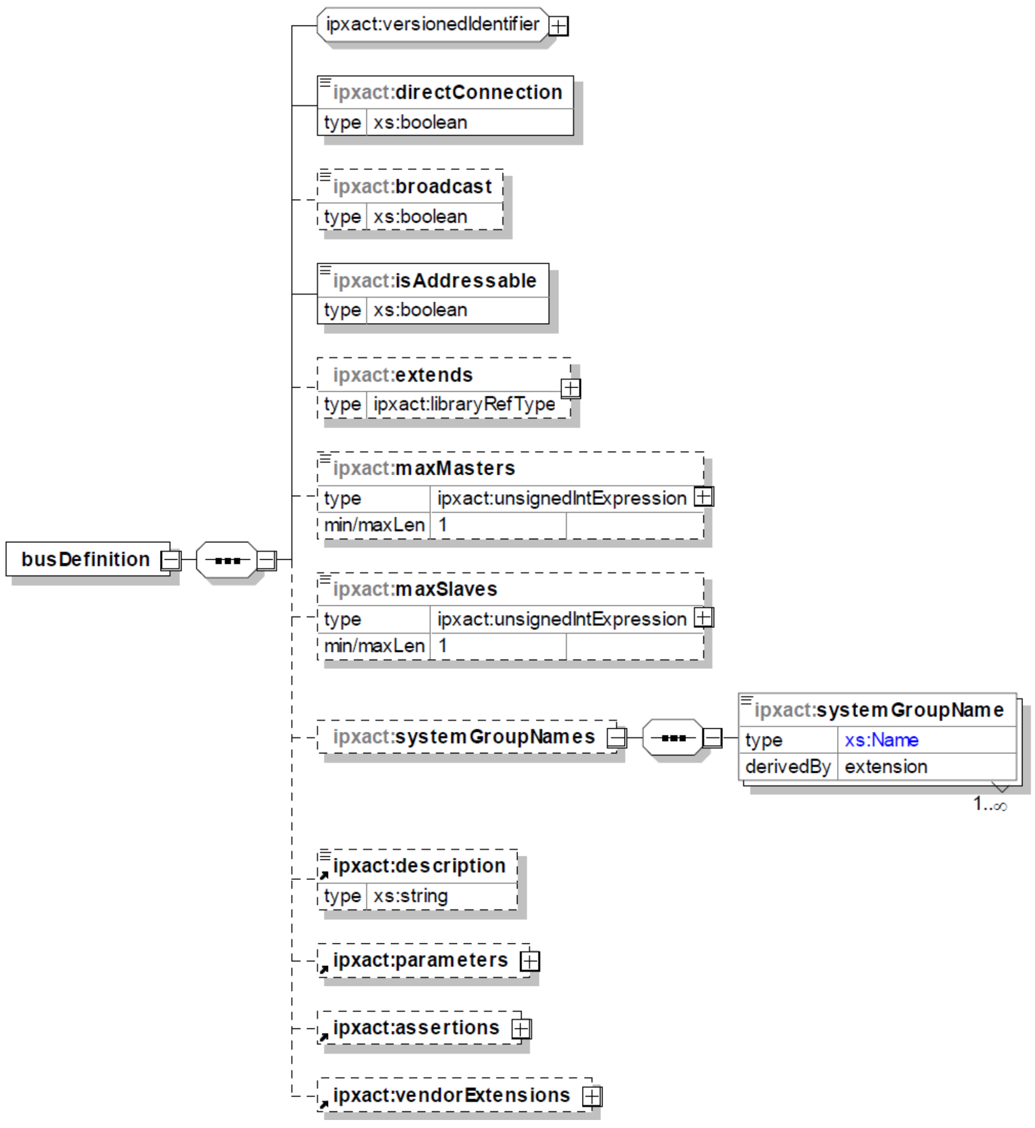
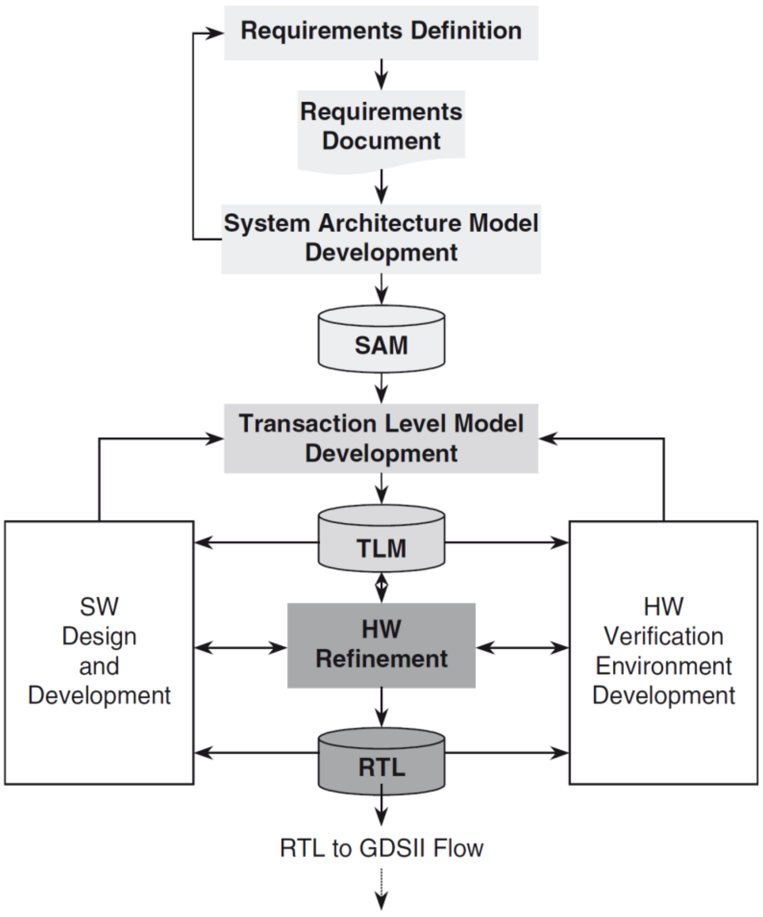


Figure 2-a busDefinition xml-element [13]

2.3 Transaction Level Modelling

Transaction Level Modelling (TLM) [19] is a way to describe an IP in abstraction levels above RTL. TLM uses function calls (transactions) to represent RTLs pins, signals and actions. Individual design teams might not see the benefit of TLM at first, but it is very beneficial as a top-down approach, especially from a system architecture point of view [4]. TLM enables architectural modelling, algorithmic modelling, virtual development platform and functional verification with much faster simulation speeds than RTL. These provide benefits as earlier software development, earlier hardware functional verification and a clear path from customer specifications to detailed software and hardware design.



Requirements Definition
Requirements Document
System Architecture Model Development
SAM
Transaction Level Model Development
TLM
SW Design and Development
HW Refinement
HW Verification Environment Development
RTL
RTL to GDSII Flow

Figure 2-b Design flow with a TLM environment [4]

Figure 2-b above display a design flow with TLM interpreted. The TLM model work as a reference model between hardware and software teams. The lowest level of abstraction is the System Architecture Model (SAM), an untimed, point-to-point model to describe the system as high-level block diagrams. GDSII is a format for rendering the physical chip layout, so the arrow pointing downwards describes RTL to chip-layout/chip production. In Figure 2-c, the RTL model equals **F**. Implementation model, while SAM is **A**. Specification model. Everything in between is considered functional TLM.

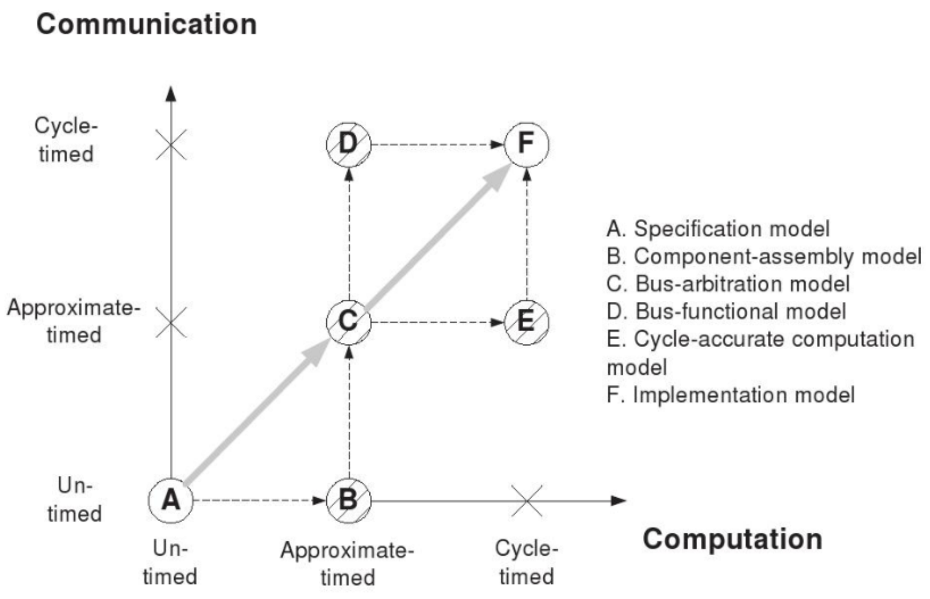


Figure 2-c Transaction Level Modeling [20]

A TLM design flow usually moves through multiple models, where each model should functionally correspond to the predecessor. To run functional verification and performance modelling one must at least be using an approximate timed model, whilst SAM is sufficient for simple architectural modelling.

TLM is generally an idea of a system that uses transactions to pass messages around. To use TLM, a programming language that has such a transaction implementation must be used. Usually TLM is a library building upon the underlying programming language. Programming languages that has available libraries include SystemC [9], SystemVerilog [21] and Matlab [22]. Universal Verification Method (UVM) [23] and Open Verification Method (OVM) [24] are the common methods of using TLM in SystemVerilog, while Matlab has Simulink [22]. The SystemC TLM library is by far the most common and complete TLM implementation to date. Open SystemC Initiative (OSCI) TLM is the name of the initial library, which now lies under Accellera [25]. The electronics industry has embraced SystemC as standard for building transaction level models, where OSCI TLM is the TLM library. As mentioned earlier, OSCI TLM is not a programming language in itself, but builds directly on SystemC. SystemC is a superset of C++, with its available open-source simulators and compilers [26]. OSCI recently released TLM 2.0 with additional features to describe sockets, generic payload and timing annotation [9].

2.4 LEON2

LEON [27] is a 32-bit microprocessor, based on SPARC-V8 [28] instruction set. It was originally owned and created by European Space Agency (ESA), now it is owned and maintained by Gaisler Research [29]. The processor core is in VHDL, and is configurable through VHDL generics. LEON2 is published under two licenses: GNU Public License (GPL) [30] and Lesser GNU Public License (LGPL) [31]. The core is LGPL, while all support files belong under GPL. In short, that makes LEON practically open-source, making it very attractive for SOC designs. LEON is originally designed for space applications, but LEON fits in both commercial and research designs. Since its creation, LEON developed multiple models, where the newer models include additional infrastructure, like a SOC. LEON2, LEON3 and LEON4 are different versions, and they often refer to SOC designs. There are also Fault-Tolerant (FT) versions of the design, but FT versions are not relevant for this thesis.

LEON2 is the second iteration of the LEON processor, and is commonly distributed as a SOC with the following Intellectual Properties (IP):

- AMBA AHB bus system
  - Memory controller
  - AHB/APB Bridge
  - Debug support unit with trace buffer
- AMBA APB bus system
  - UART
  - Timers
  - Interrupt controller
  - I/O port

Figure 2-d visually displays the LEON2 SOC with connections between each module. The implementation used in this thesis does not include the PCI/Ethernet connections nor the Debug Serial Link in Figure 2-d. LEON2 often refer to the synthesizable VHDL implementation, but this thesis uses a SystemC implementation. Magillem provides the SystemC implementation, which is not publicly available. However, it is derived from and almost identical to the implementation provided by Accellera. The implementation provided by Accellera is a part of the public IP-XACT example for LEON2 from their website [32]. The HDL implementation of LEON2 is available as Open-Source through GitHub [33] provided by Jiri Gaisler. Wherever the report mentions LEON2, it is always a reference to the entire SOC, rather than only the processor.

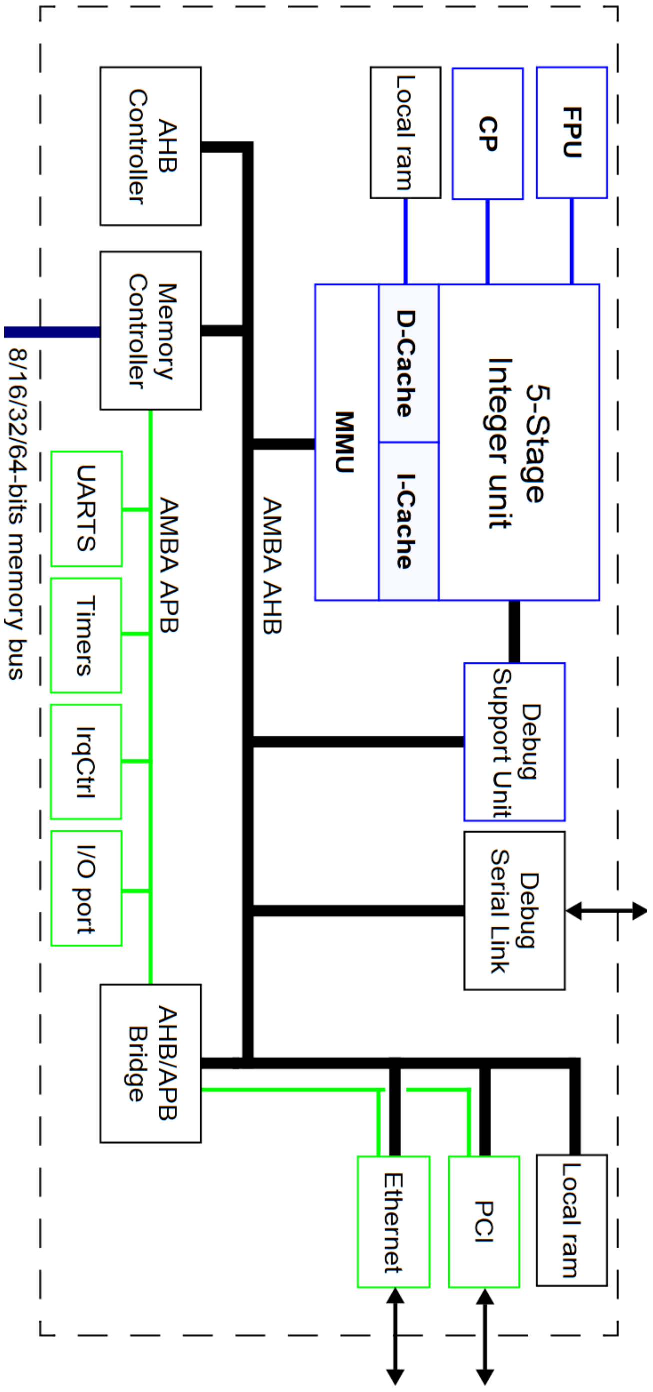


Figure 2-d LEON2 model [27]

3 Experiment

Proper investigation of IP-XACT's impact is clearer with the help of an experiment. The experiment uses a TLM implementation, but the discussion includes a very brief introduction to the RTL aspect of IP-XACT. The experiment revolves around LEON2 with its deliverables that together form a SOC. It is very convenient for the experiment that the SOC implementation is open-source and available both as TLM and RTL. Investigation consists of using IP-XACT in practice and applying changes and updates to the TLM implementation with the help of IP-XACT. The experiment will help to identify pros and cons directly related to the design flow utilizing IP-XACT.

The experiment consists of taking an IP-XACT and TLM implementation, and add a module to them. Since it is hard to get a hold of a module described in IP-XACT and TLM, the module is built from the ground up. The module must be compatible with OSCI TLM design, and then integrated with IP-XACT. Take note that an external module must match both the IP-XACT version and TLM version. There are also restraints to the module in terms of ports and connectivity to the LEON2 SOC. Connections must be to either AHB or APB bus systems, and the processor does not have extra ad-hoc connections, so any ad-hoc signals of the module will be passed outwards to the eventual test bench.

Purpose is to investigate IP-XACT's properties, not the tools' properties. However, each tool will of course present IP-XACT slightly different, thus slightly influence IP-XACT's representation.

3.1 Methodology

The entire design flow can in theory be fully described by IP-XACT in every stage. However, the experiment will only cover the beginnings of the design cycle, but further discussion will be regarding the entire design flow. In practice, a company choose how much they should embrace IP-XACT. Figure 3-a shows a simplified design flow, where IP-XACT is utilized in two places; mainly where there is need for connectivity and integration metadata.

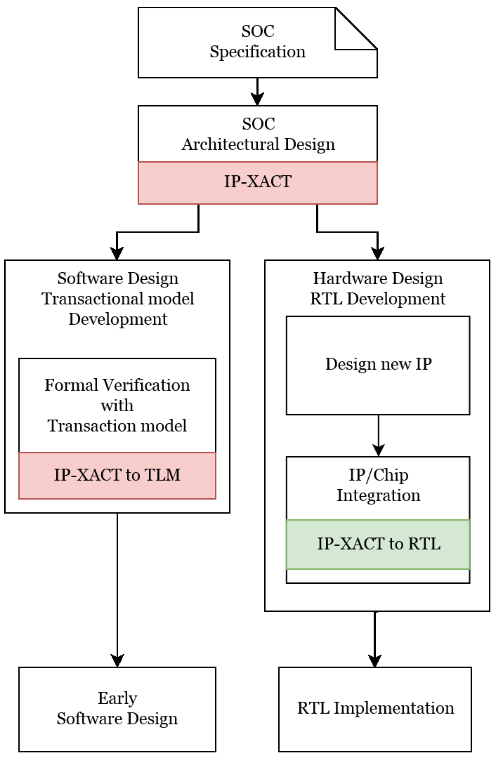


Figure 3-a Uncomplicated design flow

IP integration and the integration of the architectural design are very similar. For a design environment that already uses IP-XACT, there most likely exists a big part of the design described in IP-XACT already and either TLM or RTL. IP-XACT contributes by binding all the predefined IPs together in implementation of choice. Of course, as there are new IPs added they could practically be added to the already existing system, which is what the experiment aim to trial.

Since IP-XACT carries a more computer-aided design flow it is reasonable that what does the integration, or rather create files for the integration in another programming language, are generators. The physical generators are not part of the standard itself, but the standard include support for choosing generators as well as description of the interface between the generator and design environment. The interface's name is Tight Generator Interface (TGI) and it is utilizing the SOAP standard [34] to pass messages. All of the tools researched in this thesis both have support for custom-made generators through TGI and various built-in generators. The experiment uses built-in generators, and does not cover creating custom ones.

3.2 Setup

The setup consists of the LEON2 IP-XACT implementation alongside the TLM implementation. The most interesting areas of research here are how the design methodologies work, how IP-XACT contributes to a TLM environment, and to architectural exploration. The setup will also clarify the interactions the designer would have with IP-XACT and its impact on the entire design flow. Other areas of interest are how or even if IP-XACT stands on its own and what does the company need for IP-XACT to be effective. *Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows* is the slogan of IP-XACT, put simply that is what the experiment provides insight to.

Figure 3-b shows a visual representation of the top-level IP-XACT implementation, versions with and without RNG module are figures added in appendix A. The component with a chip illustration indicates ApbSubSystem, which implements the APB bus and the IPs connected to it. A close look at the figure reveals that the AHB bus is being passed around, where each component has either a slave or master connection. The AHB bus distribution is also implemented with its own component, which is required in IP-XACT to distribute a bus. Figure 3-c displays the internal connections of ApbSubSystem. Similar to the LEON2 model from Theory the first block is the AHB/APB Bridge (i_ahb2apb). In this figure, it is also clear how the bus distribution works in IP-XACT (see i_apb).

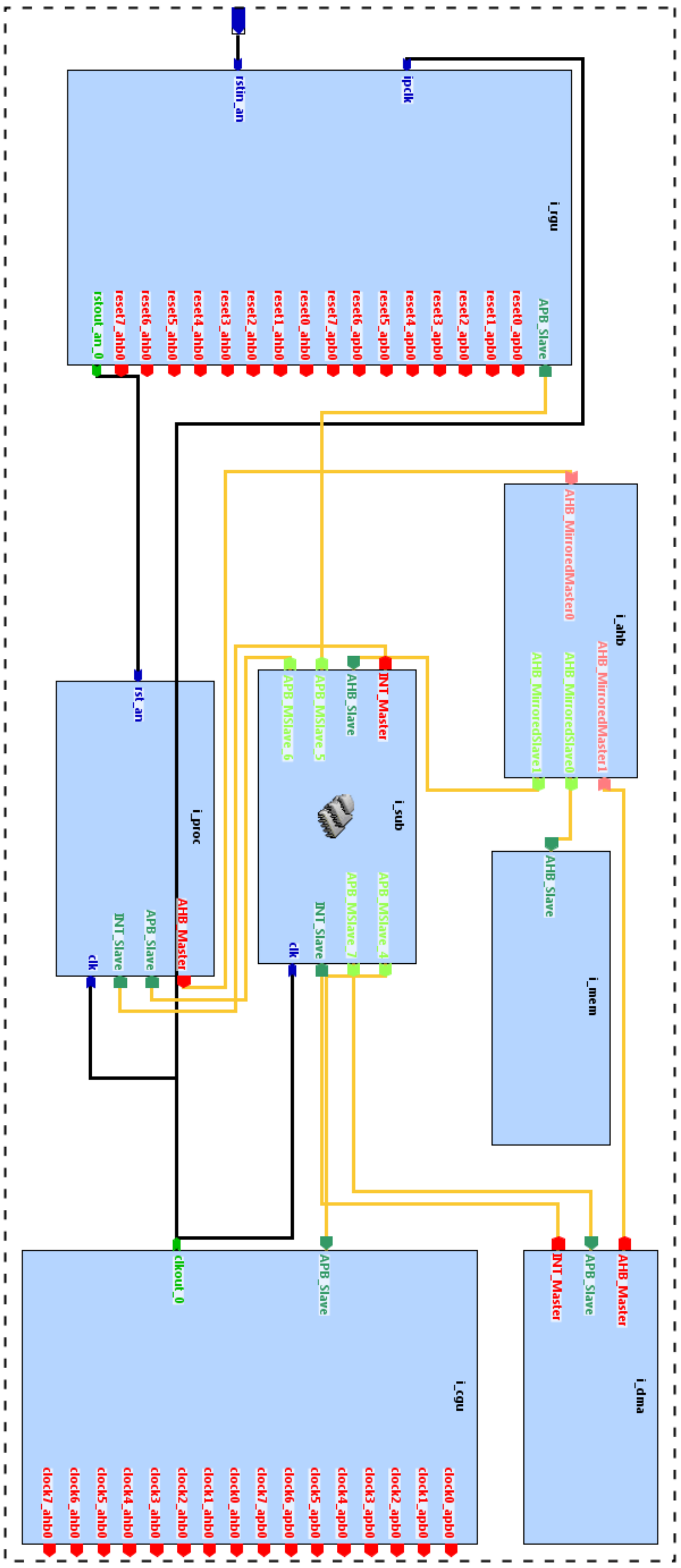


Figure 3-b LEON2 top-level blocks in IP-XACT[2]

[2] Graphical illustrations from the tool *Magillem*

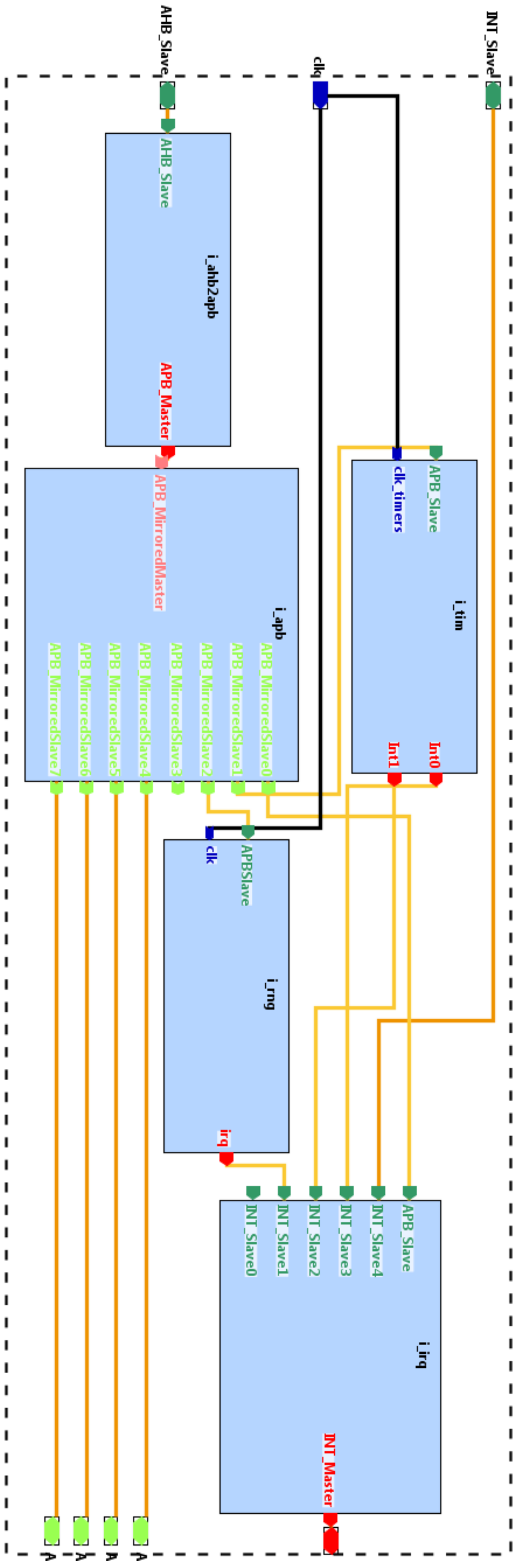


Figure 3-c Blocks within ApbSubSystem (i_sub) [2]

IP-XACT resembles a black box design, a design where there is little to no information on internal behaviour. The module needed for the experiment is also a black box, except it has to interact with the TLM for testability. RNG module is made from scratch for the purpose of experiment, but a black box module from an external provider can be used as well. If an external IP is used, it only has the constraint that it has to be compatible to either AHB or APB for easy connection. If it were not compatible, one would have to change the internals of other black boxes to connect it to the design, which is not desired.

To put the experiment into context with an industrial design flow, the designer would most likely already have a partly assembled design. An industrial design is likely to be more complex, but the way IP-XACT work and the interaction between IP-XACT and TLM is almost identical. How the designer adds a component to the design is identical, as it is necessary to use the exactly same constructs of IP-XACT. ARM's AMBA AHB/APB [35] buses are also very common in the industry. IP-XACT describes views, which in short resembles point of view. The experiment uses a TLM point of view, with no point of view configured for RTL.

The addition of a module sheds light upon how to create, add and modify a component as well as how to integrate it into the already existing design. The act of removing/moving the component should be investigated, and if its interfaces change. The interface change consists of first adding the bus connections and later add an interrupt and a clock. The act of adding or altering the design is in theory a form of architectural exploration.

Adding connectivity through IP-XACT is one thing, if the new connections fair well with the TLM/RTL implementation is another. To be able to add on extra connections after creation, the design has to be parameterized completely, but the experiment implies that the design is. An interesting feature here is if IP-XACT has constructs that supports a parameterized design in the sense that it can hold and set those parameters.

3.2.1 RNG Module

Finding an IP-XACT open-source module that fits LEON2 is hard; hence, it is reasonable to create a new module for the system. As stated previously, behavioural functionality of the module is almost insignificant to IP-XACT, because mostly connectivity applies to IP-XACT. The module added to the design for this experiment is a Random Number Generator (RNG). From an IP-XACT point of view, the module can be entirely a black box design, but the TLM requires some simple functionality to ensure that the module has properly integrated with the design.

The thesis could include a complex behavioural TLM model of the module, but that is not necessary for this study nor will it have a huge impact on the IP-XACT implementation. The TLM implementation of the RNG module has functionality that allow for transactions, signalling and an interrupt. The interrupt is a *BusDefinition* description and the clock is described as a signal, which provides a bit of diversity between the two. The module itself provides local addresses of its registers, and the top-level design provides addresses for each underlying module.

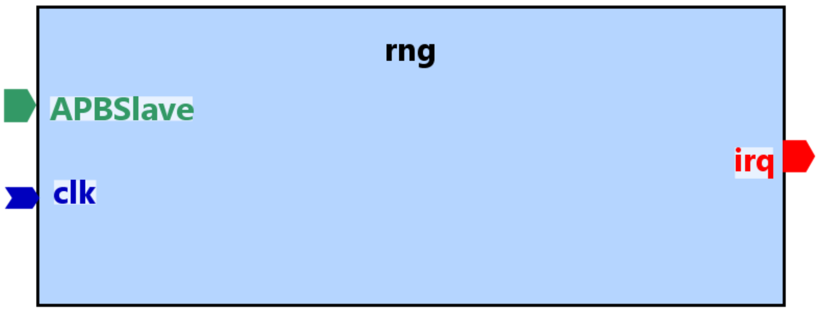


Figure 3-d RNG module

Most tools support some form of automatic generation of TLM/RTL, and often provides IP code skeletons for further development. The experiment will not emphasize the generation of skeletons, since that is not directly bound to the standard.

3.3 TLM Implementation

In the industry, iterating on products and during creation of new systems it is very common to reuse a lot of the design. This experiment reuses an implementation similar to the IP-XACT example at Accellera's site [32]. The implementation is approximately timed TLM, and belong to *B. Component Assembly Model* from Figure 2-c in Theory. Using this TLM version saves a lot of time due to reuse and simplistic design, where it should be easy to add an IP without rewriting big parts of the code. The design ensures testability from the processors point of view, where the designer can specify write and read operations to specific addresses from the processor. The implementation is in SystemC, so there are available free tools and simulation is fast. IP-XACT as a data-book for metadata role in the design flow will always be as a support flow to either TLM or RTL. Realistically TLM carries more of a support role for RTL, but for this experiment, TLM is the main flow and purpose.

The experiment describes the interaction between IP-XACT and TLM, but in order to rationalize deductions from the interaction, the TLM implementation should have a purpose. Adding a purpose to the TLM implementation enables deeper analysis of the interactions between the two. It can also help uncover what could be done with IP-XACT in the future to support more complex/elaborate modelling. The purpose of the TLM implementation is to perform architectural exploration. The task to perform architectural exploration carry over to IP-XACT; an interesting question is how IP-XACT allow for architectural exploration through generators for TLM and more importantly on its own.

Architectural exploration is a broad term; it might describe very different levels of abstract modelling. The architectural exploration often depends on technology, which has different granularities and benefits. Therefore, architectural exploration for the purpose of the experiment is narrowed down and concretized. This thesis limits architectural exploration adding, removing and editing blocks in the design and changing between two or more parallel designs. In the context of RTL design flow, all the points above are steps performed much earlier than RTL design in the design flow.

Although it excludes IP behavioural modelling, a discussion to how IP-XACT support more complex modelling through SystemC is further down in the thesis. Complex modelling is outside of the scope of this thesis, but how IP-XACT may support it is not. Complex modelling includes increasing the precision by moving further towards a cycle/pin-accurate TLM model, by adding on performance and energy modelling.

3.4 Tools

As IP-XACT is becoming a worldwide standard with notable supporters, more and more companies are adding support for IP-XACT to their tools. Many businesses claim to support IP-XACT, but the amount of commitment and actual support is varying. This chapter describes the selection of most of the available tools. It is important to remember that the research is on IP-XACT; not the tool using it or presenting it. That being said, IP-XACT provide structures and opportunities that might be or not be fully/practically compatible with a tool flow.

There exists many tools with IP-XACT support, and more are being created almost every year [36]. There are different levels of commitment among those; some provide full support to set up a connected design, while others simply include support to parse components, respectively categorized as full and partial. The following list categorizes most of the available tools with providers in parentheses:

- Full
    - Magillem (Magillem) [17]
    - Kaktus2 (Tampere University of Technology) [37]
    - DesignPlayer (EDAUtils) [38]
    - Socrates (ARM) [16]
    - GenSys (Synopsis) [39]
    - STAR (deFacto) [40]
    - Scineric Workspace (Scineric) [41]
    - IDesignSpec (AgniSys) [42]
- Partial
    - Vivado (Xilinx) [43]

There may, or probably, exist more tools than listed here, but this provide a good selection with some diversity. First that diversify them are cost, some are free; others are costly. Kaktus2 and EDAUtils are completely free; Scineric Workspace has half a year trial, whilst the rest are not free. Some of the rest are available under temporary academic/research licenses that require application. Since TLM is the narrated purpose of the thesis, it is convenient to sort out which tools only support RTL designs. STAR, Vivado, Socrates, AgniSys, Kaktus2 and GenSys are all tools that per 2017 is geared mainly towards RTL. There is a reason to use a tool that uses IP-XACT back-end as well, elsewise the designer interfaces with blocks that are not directly described with IP-XACT's constructs.

Magillem is the best suited tool to implement and research the standard. The reasoning behind is that they are one of the main contributors to the IP-XACT standardization, and the software is built with IP-XACT as its main functionality. With Magillem being built around IP-XACT it is possible to easily get a good picture of IP-XACT's constructs. Magillem is one of the two tools that add TLM integration, the other being EDAUtils. Magillem is also the better, more stable and more user-friendly tool of the tested ones. Out of the tools listed; Magillem, Kaktus2, EDAUtils and Scineric Workspace were installed and tested.

4 Experiences

The experiment is very valuable for the general understanding of IP-XACT's methodology and usage. This chapter describes observations that provide value to the discussion to come. All the experiences derive from studying the IEEE standard itself and observations throughout experimentation.

4.1 Methodology

Initially IP-XACT only described components, but has expanded a lot since then. IP-XACT is now able to describe entire systems, and to be a part of the entire design flow. IP-XACT much resembles the system architecture model (SAM) with additional properties to help the rest of the design space.

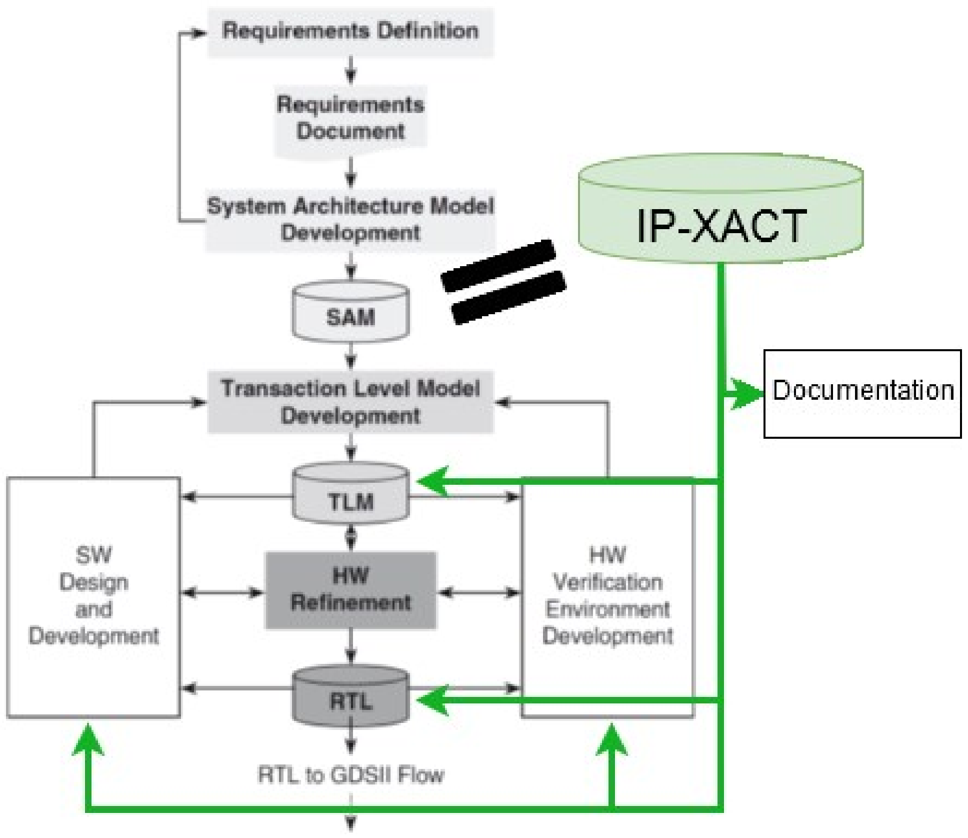

Requirements Definition
Requirements Document
System Architecture Model Development
SAM
IP-XACT
Documentation
Transaction Level Model Development
TLM
SW Design and Development
HW Refinement
HW Verification Environment Development
RTL
RTL to GDSII Flow

Figure 4-a Design flow with IP-XACT

In practice, the arrows in Figure 4-a describe generators, but for the design flow they also stand for general knowledge. Knowledge about the entire design and its connectivity is very valuable to the designer throughout the flow. IP-XACT serve as an extended SAM applicable to all parts of the design, where every part uses the same exact reference model for further development. This is a valuable resource to the design teams early in the design cycle.

IP-XACT ties all the different environments together with the help of views and file-sets. Views are usually design views, simulation views or documentation views. In the experiment the common view describes the SystemC design, simulation view describes simulation directives and dependencies, and documentation view describes documentation. File-sets specifies which files, what kind of files and what view they belong to. Names of the views must be global, while they are used locally. With IP-XACT describing views and links to different environments, development teams can all work simultaneously with the same reference model. Each team would describe their view and file-sets. The experiment describes the TLM view as *TLM_PV*, which is used for the RNG module too.

Views are one of the more important features of IP-XACT, but IP-XACT does not contain objects to globally define each view. The ability to describe each view and their properties would have been very useful to the general understanding of the system and very practical, especially in multi-team environments with different views. At the moment, views are simply strings that happen to match, and the way to find matches is to use the view specified in the top-level *design* schema and find that name in its underlying components.

Xml is human and machine readable, but IP-XACT's complexity definitely makes it less attractive to edit and more attractive for Computer-Aided Designs (CAD). For the most part, it is not practical to edit IP-XACT schemes by hand. To deal with this, IP-XACT includes Tight Generator Interface (TGI). TGI is a standardized Application Programming Interface (API) to connect generators that use or edit the IP-XACT schemas, generators allow for easy editing of the design automatically. Since IP-XACT is hard to handle directly, tools make it easier to edit using GUIs that describes the elements in XML.

4.2 Structure

IP-XACT is a comprehensive standard that captures big parts the design with most of its data, and that is a characteristic that comes with both benefits and drawbacks.

All designers start out as novices when initially starting out with IP-XACT. It is an immense standard, with a lot of different aspects and objects to understand. The XML-syntax makes it easy to understand the structure, but does not make it easy to read. Figure 4-b displays a small part of the XML behind the *ApbSubSystem component* previously marked in Figure 3-b. The entire description totals a little less than 1000 lines of code, and that is without the internal design hierarchy. This makes manually editing the files a hassle, and in reality, not practically viable. It does become easier with a tool, tools also include restrictions according to the standard's elements. That makes it a lot easier, but still there a lot of objects that may not come as easy to the designer.

```
    <ipxact:vendor>spiritconsortium.org</ipxact:vendor>
    <ipxact:library>Leon2RTL</ipxact:library>
    <ipxact:name>apbSubSystem</ipxact:name>
    <ipxact:version>4.0</ipxact:version>
    <ipxact:busInterfaces>
       <ipxact:busInterface>
          <ipxact:name>Interrupt</ipxact:name>
          <ipxact:busType vendor="spiritconsortium.org"
library="busdef.leon2" name="IntProc" version="v1.0"/>
          <ipxact:abstractionTypes>
             <ipxact:abstractionType>
                <ipxact:abstractionRef vendor="spiritconsortium.org"
library="busdef.leon2" name="IntProc_rtl" version="v1.0"/>
                <ipxact:portMaps>
                   <ipxact:portMap>
                      <ipxact:logicalPort>
                         <ipxact:name>IRL</ipxact:name>
                      </ipxact:logicalPort>
                      <ipxact:physicalPort>
                         <ipxact:name>Interrupt_IRL</ipxact:name>
                      </ipxact:physicalPort>
                   </ipxact:portMap>
                   <ipxact:portMap>
                      <ipxact:logicalPort>
                         <ipxact:name>IRQVEC</ipxact:name>
                      </ipxact:logicalPort>
                     <ipxact:physicalPort>                <
                     <ipxact:name>Interrupt_IRQVEC</ipxact:name>
                      </ipxact:physicalPort>
                   </ipxact:portMap>
                   <ipxact:portMap>
                      <ipxact:logicalPort>
                         <ipxact:name>INTack</ipxact:name>
                      </ipxact:logicalPort>
```

Figure 4-b *ApbSubSystem* code snippet

XML structure all the objects in a hierarchy, with the eight top-level schemas as the roots. Hierarchically ordering makes a lot of sense, and makes understanding a lot easier, also most components are best described hierarchal anyway. *ApbSubSystem* also has a *design* description separate to the *component* description. The design specification only describes connectivity, so if the design requires any functionality one has to create that by adding a component within the design.

IP-XACT is no longer limited to describing individual components, but can now describe entire electronic systems. It does so with the following eight top-level schemas; *BusDefinition, abstractDefinition, component, design, abstractor, generatorChain, designConfiguration* and *catalog*. Out of the eight schemas it is only *component* that describes metadata of an electronic component as initially intended. The rest describe an interconnected system of said components, as well as the design flow. The designer does not need to use all of the eight level schemas to describe a component / system. *BusDefinition, abstractDefinition, design* and *catalog* are essential to a system, respectively describing buses, implementation of buses, connections and instantiations of a design and locations to definitions, designs and components. The top-level schemas *component, design, busDefinition* and *abstractDefinition* are enough to entirely build the experiment.

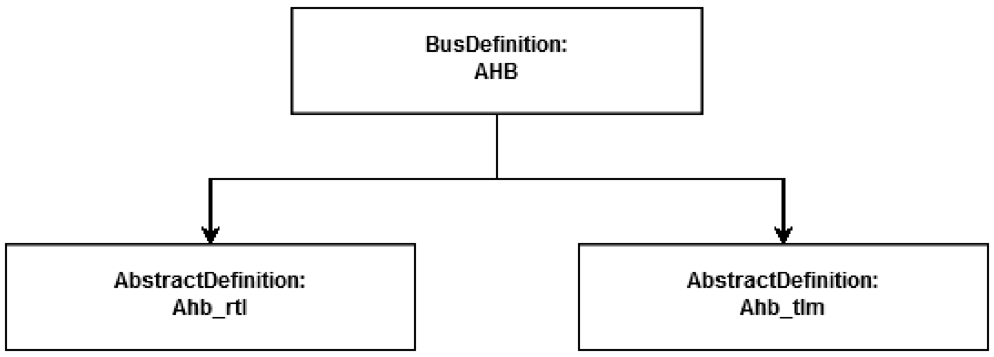


Figure 4-c BusDefinition and abstractDefinition

Implementing a bus requires two top-level schemas, *busDefinition* and *abstractDefinition*. *BusDefinition* serve as a name label referenced by *abstractionDefinitions*. Everywhere a bus is implemented specifies both the *BusType* and *AbstractionType*, where the latter holds almost all of the information. The case where this seems like it is a useful structure is when dealing with mixed designs (TLM/RTL coexist in design), where an *abstractor* connects the two abstractions. Even then the abstractions would have to be referenced, instead of only the *busDefinition*. The *abstractor* then has to be implemented within the *DesignConfiguration* element. Separate descriptions for definition and abstraction is arguably an unintuitive way of handling the bus abstractions, especially considering both schemas has to be referenced when creating an object. In IEEE 1685-2014 *busDefinition* and *abstractionDefinition* are respectively categorized as high-level and low-level attributes of a bus, but *busDefinition* holds relatively little information compared to *abstractionDefintion* [13].

There are already many elements that describes specific cases of design, with more to come in the future. This raises the complexity and learning curve, but it is a necessity. If IP-XACT is not able to describe the entire design, it is no longer viable. However, there is the addition of *vendorExtensions* that opens up the ability to describe whatever the designer or tool desires. The textbox below displays an example of Kactus2 using *vendorExtensions* to describe graphical properties within the tool. The discussion raises an interesting debate whether this is a potential clash with IP-XACT's promise of vendor neutrality and creates incompatibility problems for reuse between different design flows.

```
    <spirit:vendorExtensions>
        <kactus2:extensions>
            <kactus2:columnLayout>
                <kactus2:column name="Components" contentType="2"
allowedItems="2" minWidth="259" width="259"/>
            </kactus2:columnLayout>
            <kactus2:routes/>
        </kactus2:extensions>
    </spirit:vendorExtensions>
```

Figure 4-d *vendorExtensions* used by Kactus2

IEEE 1685-2014 IP-XACT also introduces the *protocol* element:

> *A protocol element characterizes the communication protocol. It contains a protocolType and (optionally) a payload. The protocolType can be tlm or custom. [13]*

Protocol may be added to *component, model, ports, port* or *transactional* elements. The protocol element may help to describe behavioral information related to interface communication, but this is perhaps another deviation from the limitation of no behavioral metadata and vendor neutrality.

4.3 The Experiment

IP-XACT plays a big role in the implementation of the experiment and proved very useful both as SAM and with the SystemC code generators. This chapter describes the rough lines of the IP-XACT implementation and how it relates to TLM.

As mentioned before, IP-XACT interpret an interface block centric approach. Even buses are also defined as blocks in case there are more than exactly one slave and one master connection. Bus interfaces and ad-hoc signals are not blocks, and is in practice the only things connecting the design. The bus interfaces are defined globally, first as a *BusDefinition*, then as an *AbstractionDefinition* to describe the more physical or transactional signal lines. The bus referencing methodology is very convenient because it makes it easy to distribute the same interface without explicitly specifying signal lines. It also makes it easy to switch between views, since interfaces reference the global definition rather than each module's definition. Ad-hoc signals are equal to single ports from an RTL perspective. The inclusion of ad-hoc signals simplify the IP-XACT structure, since the designer can create signal lines that does not have to be globally defined. *AHB, AHBLite* and *APB* are buses utilized in the experiment.

It does help that objects have a lot of optional elements, which can serve as advanced options for advanced users. The experiment was able to demonstrate some of the advanced features, but others were left untouched. The elements not experimented with are *abstractors* and *generatorChains*. An *abstractor* is a connection between two different views, an example would be to connect *TLM_PV* with *RTL_PV* through an *abstractor*. *Catalog* is a list of references to IP-XACT descriptions for a design, similar to a library for most programming languages. *Generator chains* are a representation of the design flow for a computer. Here it is possible to specify a list of actions to be performed on the file-sets in a *component* or *design*.

4.3.1 RNG Module

To set up a component in IP-XACT there are four things to set up: *busInterface, memoryMaps, model* and *file-sets*. *Bus Interfaces* are connections to buses that already have a defined *AbstractionDefinition* in the design; for the RNG module that signify the *APBSlave* and *irq* in Figure 4-e. Both the *BusType* and *AbstractionType* are necessary to correctly set up a bus interface, and they must correspond to their top-level definitions. Memory Maps are not strictly necessary for the design, because they will be defined in the TLM implementation semi-manually. For good practice and sake of experimentation they are added anyway.

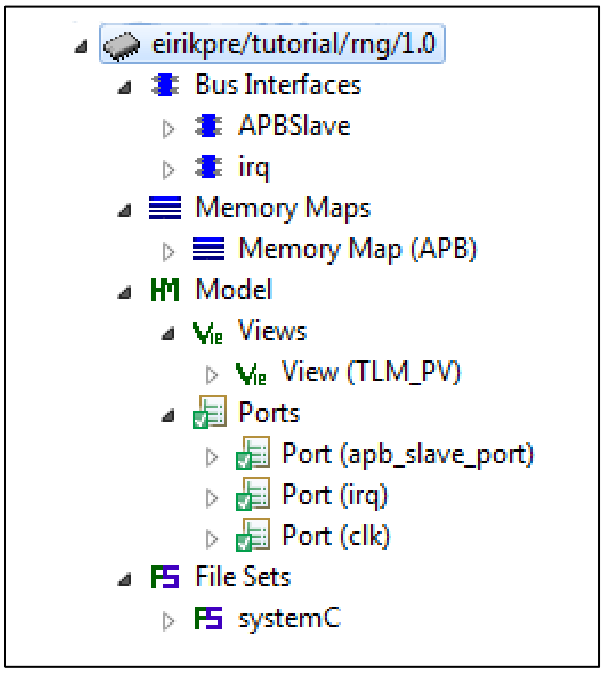


Figure 4-e RNG module IP-XACT representation

The Model is split into Views and Ports. Views are presented previously, and the module uses *TLM_PV*, which is common for the design. Ports are the physical ports of the design. If a port is bound to a bus interface in *BusInterfaces* they will be represented as interfaces instead of signals. The module binds APBSlave and irq to their respective *busDefintions*, while *clk* is left as a signal. The file-sets are links to the SystemC implementation of RNG module, file-sets describe everything related to an external file such as simple build commands and dependencies.

When connecting the RNG module to *ApbSubSystem* a couple of new entries are needed. IP-XACT instantiates modules under *componentInstances*, then connects buses and signals under another element. This makes adding and editing instances and connections very easy, especially for a tool doing the editing based on designer input. Connections differ whether they are defined buses or not; a predefined bus is an interconnection while non-defined are ad-hoc connections. This distinguish the two, and makes it easier to view a design. *BusDefinition* is linked to the ports in a component, which means if the definition change, the component has to be changed accordingly. From the component's perspective ad-hoc connections are just named ports that has not been bound to a bus interface.

```
        <spirit:componentInstance>
              <spirit:instanceName>i_rng</spirit:instanceName>
              <spirit:componentRef spirit:vendor="eirikpre"
                    spirit:library="eirikpre" spirit:name="rng"
                    spirit:version="1.0"/>
        </spirit:componentInstance>
```

Figure 4-f Instantiation of RNG module in *ApbSubSystem*

The IP-XACT description of the RNG module is in Appendix B. It is written for 1685-2009, so the namespace is *spirit* instead of *ipxact*. Magillem became the most used tool because of its ability to present data as close to IP-XACT standard as possible. Magillem does not yet support IEEE 1685-2014, hence the design is described in 1685-2009. This impacts the way IP-XACT handles transactions. Notably, 1685-2014 adds TLM 2.0 support with a more elaborate *transactionalPort* including *tlm_sockets*. The features of IEEE 1685-2009 is enough to entirely describe the TLM implementation in the experiment because of a small workaround using exports on buses and ports on single ports.

Almost all IP-XACT tools include a way to auto-generate parts of the top-level integration. Automated top-level integration helps to combine the design, especially when being less familiar with the design. After instantiating and connecting the module within IP-XACT, a header file describing the design of *ApbSubSystem* can be generated. The header is entirely plug-and-play into the SystemC design, and if the SystemC RNG module is functioning correctly and included in the TLM design, the design compiles and runs right off the bat.

```
<spirit:port>
       <spirit:name>apb_slave_port</spirit:name>
        <spirit:transactional>
          <spirit:service>
            <spirit:initiative>provides</spirit:initiative>
            <spirit:serviceTypeDefs>
              <spirit:serviceTypeDef>
                <spirit:typeName
spirit:implicit="false">OSCI_TLM_PV</spirit:typeName>
                <spirit:parameters>
                  <spirit:parameter>
                    <spirit:name>typedef1</spirit:name>
                    <spirit:value>ADDRESS_TYPE</spirit:value>
                  </spirit:parameter>
                  <spirit:parameter>
                    <spirit:name>typedef2</spirit:name>
                    <spirit:value>DATA_TYPE</spirit:value>
                  </spirit:parameter>
                </spirit:parameters>
              </spirit:serviceTypeDef>
            </spirit:serviceTypeDefs>
          </spirit:service>
        </spirit:transactional>
      </spirit:port>
```

Figure 4-g Transactional port in IP-XACT

To represent a parameterized transactional port in IP-XACT it is required to define parameters, the type of port and direction. Figure 4-g describes the APB bus as a transactional port within RNG module. The example is in IEEE 1685-2009, with the old *transactional* port version. Solely using IP-XACT one can see that it is not able to describe the type of port. To solve this, the code generator makes the presumption that if *initiative* equals provides and if it is a bus interface, *sc_export* instead of *sc_port* is used. The result becomes the code below, with *pv_target_port* being specified in *AbstractDefinitions->ahb_pv->port->transactional->onSlave->service->typeName*:

```
pv_target_port< ADDRESS_TYPE,DATA_TYPE > ahb_slave_port;
```

The experiment effectively gives two designs that can be used for architectural exploration with a purpose. If the TLM implementation supports elaborate performance modelling, a purpose could be to compare the two designs power consumption, area and throughput. One design would have RNG implemented in software on the processor, while the other design uses the RNG module to generate numbers. As before, to change between the different designs it is as simple as generating the top-level integration. That even opens up for a executable specification, where you do not keep different versions of designs in TLM, but rather keep versions within IP-XACT and update them as needed.

4.4 Model Proposal

As a response to some of the shortcomings a new model is proposed. The proposed *model* is a hypothetical restructuring to make methodology more consistent in a mixed-model design. The new structure unites all aspects of a model under one top-level schema, which then can be referenced by VLNV like *busDefinition.* The main concept that is restructured is the interpretation of *views*.

To say that *views* are being referenced in a design is not entirely true, the experiment show a case where *views* just match in the name, but are not defined or referenced anywhere like *busDefinition/abstractionDefinition* is. The wording of *Views* is also rather unintuitive and imprecise. *Views* describes point of view of a component/design, why not just name it *model* since that is what it describes. This chapter proposes a new *model* that entirely replace the old *model->view*. *Model* has the same purpose as the old *view*, which is to describe the *model*. The *model* proposal includes everything that might be used to specify the model specific details, which spans over the entire IP-XACT structure. Figure 4-h describes the original top-level methodology, where red arrows indicate potential dependencies. The old *model->view* lies within the *components* element. Top-level schemas are interpreted as separate sheets, referenced by VLNVs.

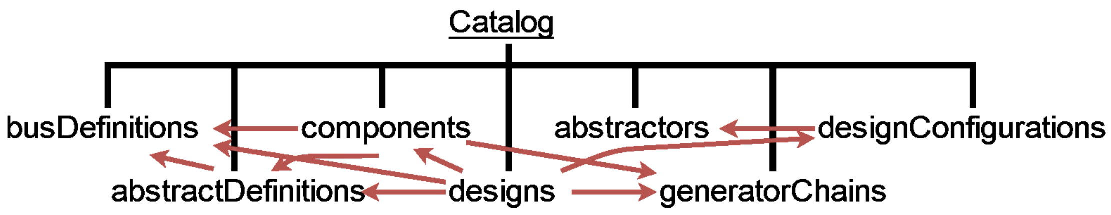


Figure 4-h *Catalog* element

An elaborate mixed-views design will have a lot of dependencies back and forth, which makes the design very hard to control, especially with more designers playing with it. Even in the experiment it were hard to grasp what is used where, and how it is all tied together.

The first thing the proposal change is reducing the double-layered bus methodology to only one layer. The idea of an abstract bus implementation with abstractions of said bus still sticks, but they will be moved into the same group *buses* (from *catalog* perspective). Figure 4-i visualise the one layered bus, with AHB *bus* as the example. The wording of XML-tags is not important, so it may keep the old name *busDefinition*. However, high complexity is an argument of the analysis, thus we prefer a rather simple name instead of a long cluttered name; *bus*.

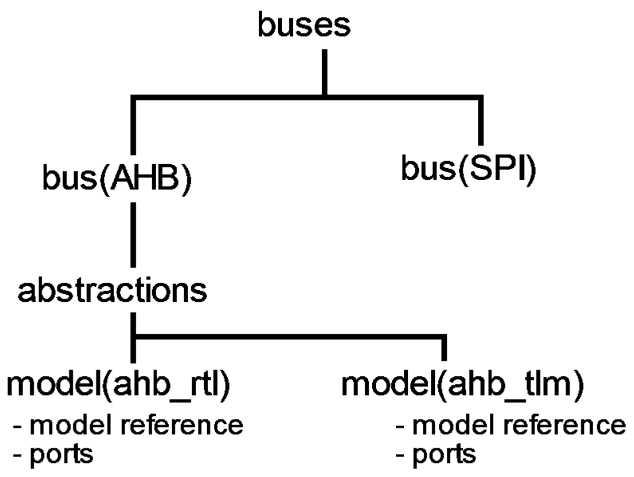


Figure 4-i Proposed *bus*

The second proposal is more impactful to the entire system. The second proposal is a top-level schema named *model*. *Models* describe everything related to a specific model (e.g. TLM or RTL). There are already top-level schemas related to models and their interoperability; *abstractors, designConfigurations* and *generatorChains* are all describing something specific to a model, thus these can be moved from separate top-level schemas to *model.* This makes the overall methodology less cluttered, more intuitive and easier to understand. Figure 4-j shows the new *catalog* with *models* and the dependency pattern in red. The figure also uses buses according to the proposal above.

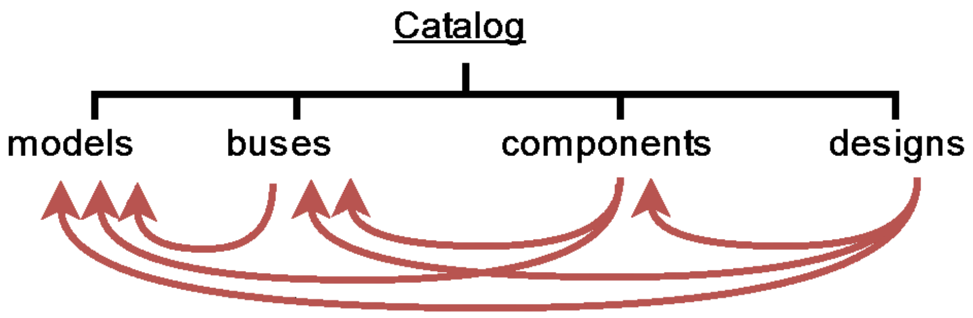


Figure 4-j *Catalog* element with *models* proposal

One of the very robust methodologies of IP-XACT is in fact *busDefinitions*. *Models* works in similar fashion to how *busDefinitions* are distributed down through the design. The new top-level methodology form an intuitive dependency pattern described by a pyramid in Figure 4-k.

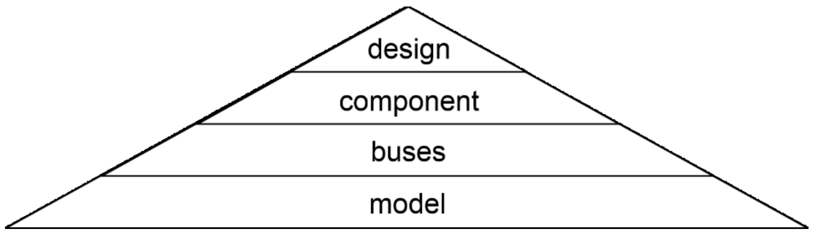


Figure 4-k Dependency pyramid

The *model* holds every element that previously described model specific details. It may also include *file-sets* and *compatibleModels*; see Figure 4-l. *File-sets* is one of those features that make IP-XACT shine, it can in the *model* element be used to describe documentation regarding the model. *CompatibleModels* solves a problem presented above where a *view* uses a name slightly different from the rest of the design, but describes the same model (e.g. TLM, tlm or TLM_PV). In *compatibleModels* one list all those *views* (now *models)* by reference, so component can be linked to the current top-level *model*, even though using an external *model*.

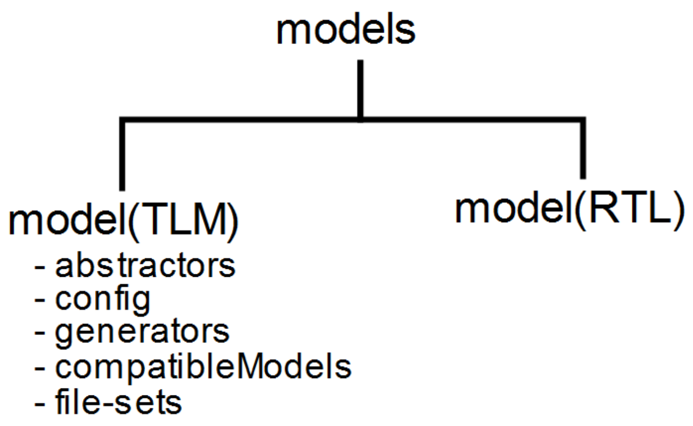


Figure 4-l *Model* proposal

The *model* proposal can of course have a lot more elements under it, but the figure above describe a clean and clear prototype. The *model's* dependency pyramid contribute to a more top-down approach with the model as the core. The goal of IP-XACT is to describe metadata of electronic components, which in term is described as implemented models, hence centring IP-XACT around the *model* is practical and in line with the scope of IP-XACT.

With a more elaborate *model* scheme, a small change to *component* must also be considered. Figure 4-e describes the current RNG module with *ports* as a separate element under *model*. To keep the *model* methodology consistent, the specific *ports* now belong to a specific model. For example, Figure 4-e show the bus *apb_slave_port* under *ports*, although it is specific to the TLM model. *Ports* separate of *models* is a less scalable approach since the type of ports would define what design they are used in. A quick example of a hazard is if UVM (hardware verification) uses 3 ports, while RTL uses additionally 7. Since they are both hardware ports (*wires*), there is no way to easily decide what ports belong to the current *model.* The new *component* holds *ports* according to the *model* element according to Figure 4-m.

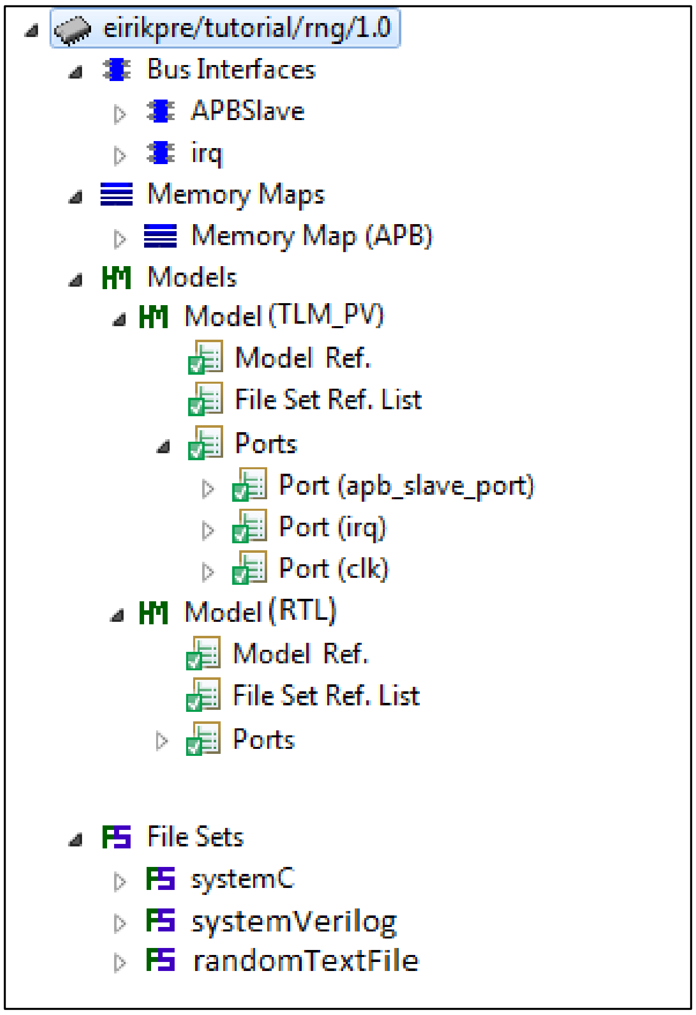


Figure 4-m New *Component* structure with *model* proposal

5 Discussion

This chapter aim to clarify and debate IP-XACT's useful features, draw backs, potential solutions and general viability. IP-XACT is still continuously being developed, so clarifying the above will hopefully provide useful feedback and help future development. The Discussion first debates the scope, methodology and structure of IP-XACT and industry perspective with emphasis on implementing IP-XACT design flow. At the end, it discusses future aspect of IP-XACT and the *model* proposal that solve some of the weaknesses possessed by the standard.

The experiment gives valuable information regarding the structure and methodology. The experiment is a relatively simple construct, yet it provides great knowledge of IP-XACT and its methodologies. An experiment in a TLM environment provides all the benefits of TLM, especially rapid prototyping through simple implementation and quick simulation. The methodology of IP-XACT in an RTL environment is very similar to TLM; arguably there are no drawbacks related to picking TLM before RTL in terms of IP-XACT representation. Also, the fact that Accellera provides open-source implementations for RTL and TLM is very convenient.

5.1 Scope of IP-XACT

IP-XACT describes metadata in and about electronic components. The components may be analog or digital, hardware, software or abstract, hard-IP or soft-IP; to describe all kinds of deliverable components IP-XACT has to contain a lot of structure for content. Initially IP-XACT only described individual components, but have grown to describe entire systems. Such an immersive standard raises both the use cases and the complexity. The scope targets CADs, and not a manually editing of the source files, thus the complexity is less important, since tools presents the information as readable and manageable as possible. It is important to note that the scope does not include behavioural characteristics of components / designs.

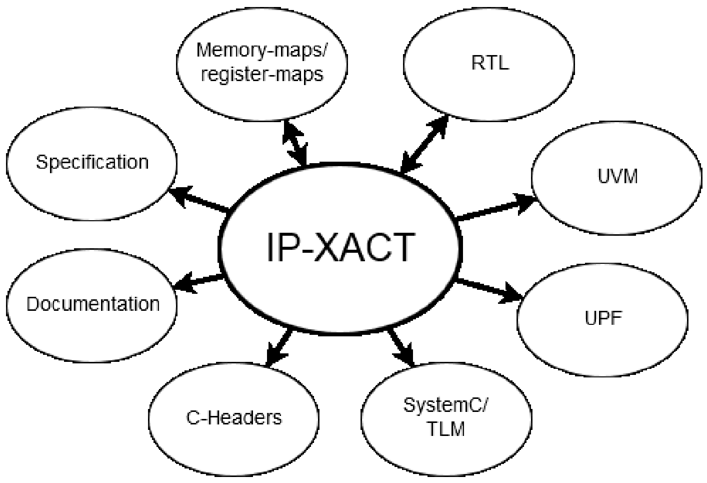


Figure 5-a IP-XACT Influence to other implementations

IP-XACT takes an expanded approach, which may or may not pay off. An expanded approach means it expands to the system around an electronic component, such as documentation, register tables and C headers in addition to the system's source code. It expands the use cases for IP-XACT by a tenfold, whilst raising the complexity equally. Figure 5-a attempts to describe some of the use cases for IP-XACT. The arrows signify how IP-XACT usually interact with the different cases; IP-XACT is meant as the source of the system design, hence most of the arrows point outwards. The arrows pointing the other way describes generation of the setup IP-XACT, which from then on should be used as a file-set reference. Unified Power Format (UPF) [44] and UVM [23] are related to hardware implementation.

It also expands the design flow, Figure 4-a Design flow with IP-XACT describes IP-XACT equal to the SAM and it being used as a reference model throughout the entire design. The scope of IP-XACT extends to being a reference model which can be used functionally to generate integration parts of the design, in addition to describe metadata of components from start to finish in the design cycle.

Metadata is a relatively loose term, which raises the question; what metadata should IP-XACT include? Since it describes electronic components at the very core it is only natural to include them. It expands to electronic systems, which is also considered here as the core of IP-XACT. Since it has taken an expanded approach both for the design flow and the influence to other implementations, the amount of metadata piles on. They both apply to the design flow, but one is more essential than the other. Influence is key to the viability of IP-XACT, without being able to provide value to other implementations IP-XACT would not be more than a passive connection diagram. The influence should be weighted to fit with the core of IP-XACT, in electronic systems the aim is to produce a physical design, thus IP-XACT should target metadata for the physical design properties. This point coincides with next statements regarding the tool flow. IP-XACT attempts to describe the tool flow, and store information regarding how tools should handle the metadata contained within. The tool flow is also custom to each company using a different set of tools, which violates the property of vendor-neutrality. The tool flow is related to the electronic component, but it is in theory not a part of the components, but their use in tools. Chapter 5.3 concretizes this claim with the example of *generatorChains*.

XML provides great flexibility and schemas can easily be appended to, thus hierarchical depth and complexity of IP-XACT grows equally to the scope without hindrance. XMLs hierarchical take on information make it very easy to realise the scope with extensive use of new tags and optional elements. There is a line between being useful and being excessive; fortunately, IP-XACT can safely expand with optional tags without altering the required core of IP-XACT. Although information is optional, does not deny the fact that it raises the complexity of IP-XACT.

5.2 Methodology

IP-XACT unites the design-flow and tie development teams to one reference model. In complex systems this carries great value, since there is likely to be multiple different versions handled independently by each development team. This is created by the coexistence of views and file-sets. The united design flow also benefit from the characteristic to not describe behavioural data; that allows for each view to specify their behavioural data without exposing it to the reference model.

IP-XACT is an interface / black-box design, which makes it easier to use as a reference model. The black-box design and hierarchical properties of XML make IP-XACT less prone to scalability problems because complexity added to one element is likely to not be directly visible in the parent element. With the technology and industry both growing exponentially, having good scalability is very beneficial for future prospects. The references to other definitions / components work better in bigger designs with the Vendor Library Name Version (VLNV) identifier [45], which makes duplicate entries nearly non-existent. VLNV is very relevant for IP-Reuse, as every IP may be referenced easily and uniquely.

The expanded *design* methodology of IP-XACT provides more information than just metadata. The *design* describes interconnections in a design, and with it the designer can create a representation of the complete model within IP-XACT. The *design* is further used to generate top-level integration files in TLM/RTL. *Design* adds a lot more purpose to IP-XACT, and effectively makes IP-XACT useable as a functional SAM.

5.3 Structure

The concept of globally defining buses is a very useful feature, but uncertainty is tied to whether this is the best way to interpret coherent abstractions. The experiment gave no evidence of the use of data contained by *busDefinition*, other than the partially useless feature of checking if *BusType* in *component* corresponds to the *abstractionType's busDefintion*. It is without merit because it has to be the corresponding type that is already defined in the top-level schema anyway, and there is no information contained in the reference other than the name. Bus definitions introduces a problem to the naming consistency of the VLNV system. If a company writes their of definition of e.g. SPI with the VLNV *company:leon2:spi:1.0*. Then the company buys an IP from a component vendor with the SPI bus *componentvendor:chip:SPI:1.0*, they are not compatible even though they are identical in implementation. Companies like to brand their own products with their own name, rather than the competition, so co-existing standards are likely to exist. Accellera provides a solution to the problem by predefining many common bus system, but cannot cover them all.

Spirit Consortium chose XML as the language to describe IP-XACT. That limits the scope to only describing the metadata, rather than doing anything functional with it. That raises the question if IP-XACT should even attempt to describe functional behaviour, or restrict metadata to electronic components only. IP-XACT effectively has implemented a kind of functional behaviour through the means of *GeneratorChains*. *GeneratorChains* describe tool chains for the entire system or for individual components (files linked in the components). Tool chain for the TLM implementation is equivalent to compiling and linking, and includes input commands and parameters et cetera. First off, this conflicts with IP-XACT only describing metadata, since it indirectly describes functional behaviour of the design flow. Second, it describes information specific to a custom compiler/tool which in term, is specific to a custom computer system or program. Vendor neutrality (also machine neutrality) is an important feature of IP-XACT, thus the addition of *generatorChains* seems like a contradiction to the scope. The pro of having a *generatorChain* is more evident if a design/component stay within only one system, since the designer can specify how to totally automate application of the *file-sets*. Even though *generatorChains* are practical in regards of a CAD, IP-XACT should arguably be careful when expanding too much in this direction, as it makes IP-Reuse and vendor-neutrality harder to maintain. Luckily, as almost everything else, they are optional to a design. *GeneratorChains* is the top-level schema, but there are also the *componentGenerator* element found within every component to further complicate the tool chain methodology.

Designs and *components* IP-XACT add something very useful, file-sets. File-sets are references to the design files. In the experiment that would be references to the belonging SystemC file/files. The file-sets allow IP-XACT to properly function as a core specification with references to all of the separate design teams. The RNG module could reference the TLM implementation, RTL implementation, netlist implementation, documentation and RTL test benches. Gathering all of this information in a central place is essential to IP-Reuse and as a consistent reference model. For example if all of the models above were included in a delivery from a component vendor, the tool could use IP-XACT to easily deduct and use the correct file for a desired implementation.

There is a lot more to the structure than discussed in this chapter and in Experiences, but there is one more interesting structural feature to look at: *vendorExtensions*. *VendorExtension* is a tag which can contain whatever the designer desires, and it can be added to any object of the IP-XACT standard. This is potentially conflicts entirely with vendor neutrality. The Kaktus2 example in Experiences shows a harmless case, since it is not used to change the physically properties of the module in any way, but it might be used it to describe information regarding a component's ports, interconnections et cetera. The possibilities are endless with *vendorExtensions*, hence for a company they might prove very valuable. From a distributed universal standard point of view, they might create several conflicting versions of the standard, which might be incompatible with each other [45].

5.4 Vendor Neutrality

To become a universal worldwide standard IP-XACT cannot be tied to specific vendors nor tools. There are a couple arguments above that clash with vendor neutrality, namely *generatorChains* and *vendorExtensions (VE).* There are also positive arguments that keep the promise of vendor neutrality. Firstly, both *generatorChains* and *VE* are optional. If a design is delivered using those, it is still possible to use the core design without them. However, with freedom comes responsibility; it is up to the designer to not specify any information that changes the component within *VE*. This unravels a potential problem where the designer can, for example, add an additional port inside *VE*. *VE* is on the other hand very crucial to the development of IP-XACT, it allows a design to function even though IP-XACT is not able to capture all the metadata required by the designer. *VE* can also be used as a prototyping platform for future elements [46].

GeneratorChains are strictly not vendor neutral. Their very nature is to describe commands and parameters specific to a tool. *GeneratorChains* is also attempting to describe a tremendously comprehensive task [47], such as Electronic design automation (EDA) toolchains and compiling/linking for programming languages. There already exist a handful of tools that does these jobs specifically (e.g. *Autotools* [48], *CMake* [49] for C), and doing so is still complex. *GeneratorChains* arguably also describes behavioural metadata, which is outside the scope of IP-XACT. *GeneratorChains* has benefits if the design stay within a design team, since it opens for a greater degree of automation.

Besides the above, IP-XACT does a good job of keeping the standard vendor neutral. The tagging system of XML in practice enumerates a lot of the content, and denies any deviation from the standard. IP-XACT also enumerates a lot of content within the tags as well, e.g. *initiative* in a transactional port must be either *provides, requires, both* or *phantom*. The namespace *ipxact/spirit* also make it clear what elements belong to the standard and which does not.

A place where neutrality might be broken is the API, where each tool could have created custom ways to interact with the IP-XACT description. Instead, the standard includes a description of Tight Generator Interface (TGI) that defines the API between the tool and custom scripts. TGI contributes to a more universal methodology, where scripts created for one tool is also compatible with another. TGI is an elegant solution to a potential hazard; it keeps the tool neutrality without implementing anything, but only defining an identical interface for all tools.

5.5 IP-XACT TLM Support

There are two main characteristics that represent TLM and SystemC. These two are SystemC ports and TLM sockets. TLM support is present in a *component's View* and in *abstractDefinition*; where it is possible to add transactional ports. The *transactional* element has received a major rework with IEEE 1685-2014, which is the version being discussed in this chapter.

IEEE 1685-2014 added structural changes that support *tlm_port*, *tlm_socket*, *simple_socket, multi_socket*, and *custom* port-types. Transactional ports can now also be properly parameterised from *typeParameters* in *transTypeDef* element. The experiment did unfortunately not include the newest implementation, but the experiment still got to include a fully functional parameterised bus export. IEEE 1685-2014 cover most reported problems related to lacking TLM support [50]. IEEE 1685-2014 also includes an overhaul to parameters, where they can now be propagated throughout the entire design hierarchy. Parameter can also be specified in all the top level schemas and parameters can be used to instantiate different elements. Parameter support is necessary to integrate a flexible TLM design.

Code generators makes creating new chip integration a nearly effortless task, which makes uncomplicated architectural exploration a breeze. IP-XACT can also serve as a graphical architectural exploration tool with checkers to ensure that connectivity is in order. TLM then serves as the next step of architectural exploration, delivering functional and timed models.

All of the necessary constructs of TLM 2.0 is already in place with IEEE 1685-2014. More elaborate modelling is probably out of the current scope, but power and area estimates might be practical for future elaborate modelling. The working group is working on power and area additions to the *component* element, which could in the end be used by TLM even though they are not specific to TLM. There is also the argument these are natural inclusions to a component's metadata, since it is a prime example of metadata, comparable to a documents size, page format etc. A more structured way to define the *view* TLM would be very helpful to a design implementing TLM; especially since designs with TLM is likely to have other *views* too.

5.6 Gain versus Cost, Industry perspective

Industry adaption is eased by the fact that IP-XACT is highly hierarchical, with a lot of optional inclusions. IP-XACT carries a greater value for bigger companies with complex designs and many separate design flows. Smaller designs should also be able to benefit from IP-XACT with a structured way to store metadata. The use cases of IP-XACT are plenty, but it is quite the feat to fully integrate and understand all aspects of IP-XACT in the work flow. The entirety of IP-XACT is most likely easier to comprehend if it is integrated in the design flow step by step. The industry should start with the core *component, buses* and *design* methodologies, and then expand from there.

A company is probably better off if they embrace IP-XACT to a greater degree, and keep IP-XACT as the only source of metadata. The transition from an existing design flow to IP-XACT is relative how much the existing design flow is integrated in an in-place CAD. If there are already a lot of scripts using the in-place metadata, there is likely to be a lot of work to rewrite most scripts, but now all of the scripts will at least be focused solely on IP-XACT.

A universal standard to unify design flows is currently non-existent. The concept of IP-XACT as a reference model for every separate flow is very beneficial to most companies, especially for bigger companies running a plethora of flows. A common reference model can ease the need for unnecessary communication between design environments and create a direct link between implementations in different domains.

The experiment implements a component from scratch similar to the industry going from non-CAD integrated metadata description to IP-XACT. Due to the black box design it is a relatively quick and easy task even for novices to IP-XACT. When converting to IP-XACT from a non-computer-aided metadata methodology, there is the need to create *components*, *designs* and bus definitions within the company or from existing libraries. Observations throughout the experiment made it evident that IP-XACT does have a learning curve associated with it, and it is necessary to understand the standard to use it. If a company decides to fully embrace IP-XACT that would require every designer to understand IP-XACT. Teaching employees IP-XACT costs time and effort. Often there is already a lot of work for an employee to learn the necessary tool flows. Tools with an in-place, practical and quick system to support IP-XACT would definitely add to the gain and diminish cost of potential learning process.

There is also the argument that IP-XACT is not really relevant to the final implementation. Theoretically speaking, in terms of quality, there is only RTL and netlist implementation that matters in the end. A designer's expertise should be on the actual implementation, rather than the systems around it. IP-XACT's benefit is more subtle, a united and structured flow contributes to the consistency and time-to-market of all design flows. In terms of knowledge, learning sufficient IP-XACT is likely only a fraction of the time it takes to really get accustomed to RTL.

5.7 Tool support

A universal standard requires it to be present everywhere. For IP-XACT to be relevant to every aspect in the design flow, every relevant tool in the design flow must use IP-XACT. Currently, that is not the case because of three problems. Tools either do not support IP-XACT, only support instantiation of a component (e.g. no support for *design*) or they support different versions of IP-XACT. The first one is easy to explain; without IP-XACT all metadata must be entered manually. When the tool only support a sub-set of IP-XACT, one cannot use IP-XACT to its full potential and there is still manual parsing of metadata. An example of this is *Vivado IP Integrator*, where it mostly uses the *component* concept of IP-XACT.

IP-XACT is relatively new to the industry and the industry struggles to keep up to date with the newest standard. This leads to disparities between which versions of IP-XACT are supported by tools. The fact that IP-XACT is not backwards compatible adds to the problem. For example, the XML namespace <*spirit:…/*> changed to <*ipxact:…/*> with IEEE 1685-2014, making newer versions completely incompatible with preceding versions.

Ideally every tool in the flow should get all their metadata from the IP-XACT model. For that to be possible, every tool has to support the same model, version, and be able to extract sufficient information so the amount of manual work can be minimized. Tool support is at the moment not sufficient and IP-XACT's expansion is very much dependent on tools since it is aimed at a CAD environment.

5.8 Model Proposal

IP-XACT's strength lies in its ideas and methodology. The sheer size of what IP-XACT tries to cover makes the complexity and depth harder to cope with. The execution of said ideas and methodology are hard to interpret in a user-friendly way. A better, more intuitive introduction to IP-XACT could make it more attractive to the industry. The working group is working on easing the introduction to IP-XACT by creating a user guide as of May 2017 [51]. A user guide would have accelerated the experiment a lot, and is probably very useful to anyone adopting the standard.

Global definitions and referencing those is an important feature of IP-XACT. *Designs* uses references to *views, components and buses*, while *components* reference *views and buses*. Buses are described in a double layered fashion with *busDefinition* and *abstractDefinition*. In the experiment there is little to justify the need to describe *busDefinition* and *abstractDefintion* separately, unless the mixed-*view* methodology is changed in the future. TLM is one of those *views*, and a proper definition of the current model/*view* was lacking during the experiment. IP-XACT describes metadata, thus it is not created to stand on its own but rather be metadata about other models. The representation of models should be more concise in the design, especially if models sport as many as in Figure 5-a.

The *model* makes it more intuitive by creating a more consistent scheme to handle *views*. One of the big pros of XML is to hide a lot of complexity in a hierarchy, but IP-XACT does not utilise that when presenting *abstractors, designConfigurations* and *generatorChains* as top-level schemas. Additional complexity at top-level clutters the standard, thus creates a less attractive first impression. The three also often end up in different files, which might in turn be scattered in different file-locations as well. The dependency pattern is an additional consistency that comes together with the model proposal. Consistency may help both the designer and the tool, as the three schemas above does not need to be explicitly referenced everywhere, since they all belong to one model that is distributed across the design.

The new *model* methodology is an alternative structure as a response to observations of the experiment. A *model* methodology will make it easier to deal with IP-XACT's internal methodology and make it more applicable to mixed-model designs. The *model* interpretation also uses the same interpretation as *buses*, which provides higher levels of consistency across internal structures. As mentioned before, IP-XACT is still a subject to development, thus a different execution of the standard's ideas might be very valuable for future development. The model still contains the criticised *generator* element, potentially clashing with vendor-neutrality, but at least now it is all contained within an element which directly relates to the specific design implementation.

5.9 Future Predictions

IP-XACT has no competition from other standards regarding metadata. With IP-XACT being the sole option helps the industry to converge on only one standard, and promote the universal aspect of IP-XACT. However, IP-XACT is in its current state no perfect solution, thus healthy competition could perhaps help the methodology by showing different takes on metadata. Healthy competition could also help push the concept of a core SAM with metadata, to really grasp the industry's attention.

How much IP-XACT is used currently is hard to analyse correctly, since companies seldom publicly share information about their design flow. That aside, there are big companies that use IP-XACT and many tool vendors have chosen to support (at least partially) the standard [52]. The more IP-XACT grows, the harder it becomes to include it in tools, especially since a tool vendor cannot/should not release unpolished products.

Three interesting features of the newest revision of IP-XACT include parameter propagation, conditionality and view-specific port maps [53]. The three will not be discussed in detail, but together they make mixed-view designs more useable and consistent. From a TLM perspective, a bulk of features has already been included in the newest version. To further support a TLM methodology IP-XACT should improve the methodology around views, so TLM, RTL and other views can interplay well in the design.

If the industry continues to follow Moore's law, there has to be systems to tackle the growing complexity and size of designs. IP-XACT propose a solution of raised abstraction and emphasis on reuse. A methodology that unites all aspects of the design flow is likely to be very valuable in the future as multiple designs flows related to one component are likely to increase. With a growing complexity and size, the need for a united metadata approach might be necessary in order to keep the design space together in the future. It might also call for IP-XACT to raise its complexity, which might make it harder for tools to interpret IP-XACT. For example, conditionality is one of the new features that brings burden to checkers, since there are a lot more cases to look for [54].

To say that IP-XACT is the sole metadata standard is correct. However, there have previously been attempts at serving the same purpose. Electronic Design Interchange Format (EDIF) [55] is a vendor neutral format to store electronic netlists and schematics. EDIF sported a vendor-neutral design methodology with flexibility. The flexibility lead to EDA vendors creating their own *flavours* of the standard, which in term made it vendor-specific. It was boosted by the fact that companies fought for market share by not creating functional EDIF writers with their tools to keep the customer tied to them [56]. EDIF tried to solve the problem in the next version by extensively expanding the standard to cover all special cases. This led to EDIF being very hard to parse and almost no tools could or bothered to do it correctly. EDIF declined in the end, due to a mix of non-user friendly elements and lacking vendor-neutrality. EDIF stands as an undesired example of IP-XACT potential future.

IP-XACT only applies to CAD, thus very much depend on tool and vendor support to expand. Chapter 3.4 Tools describes numerous tools from big companies and smaller ones, and all of them use IP-XACT in a vendor-neutral way. If Moore’s Law holds true for the next decade, future is most likely to bring more complex and bigger designs, thus designing more by hand is not likely [57]. To raise the productivity of designers, automation must play a bigger part of design flow. Automation is commonly earned through CAD and the amount of CAD work compared to manual work is likely to shift towards CAD in the future, which probably promote the use of IP-XACT. The need for automation might move metadata representation from a practical feature to a necessary feature, which probably accelerate the need and usefulness of IP-XACT incrementally. There is really no way to precisely tell what the future has in stall for IP-XACT, but the industry is definitely more receptible to such a methodology than it was when EDIF tried 30 years ago.

6 Conclusion

IP-XACT is an interesting and evolving standard that shows great promise. It provides a solution to a rising need of computer-aided design integration in the current industry, with emphasis on accelerating the design cycle. IP-XACT stands as the sole competitor to describe metadata of electronic components, which helps the standard to become broad and universal. The standard is growing positively, albeit harder to implement, many tool vendors are adding support for IP-XACT as time goes. With metadata entirely captured with IP-XACT, it can serve as a functional SAM which provides all backend metadata and references to other models. A common back-end reference model is also a valuable resource that can unite separate design flows. IP-XACT is best applied to a computer-aided design flow, hence it is very reliant on sufficient tool support. As a universal standard to all tools and companies, it is important for IP-XACT to ensure vendor neutrality. Experiences show multiple constructs that potentially undermine vendor neutrality, which makes it up to users and vendors to stay vendor neutral. Vendor Neutrality is important to IP-reuse, because without it, a model cannot be passed neutrally to every tool in the design flow.

The strength of IP-XACT lies in its ideas. Execution of said ideas become harder and harder with the extensive amount of metadata IP-XACT spans, especially as the scope of IP-XACT seem to be widening continuously. The complexity of IP-XACT makes it less appealing to implement full tool support and harder to use it to its full potential. The *component*, *design* and bus representations are all intuitive and easy to use, but most of the structure of the *model/view* implementation is a case of the opposite. IP-XACT fully support TLM with the newest version: IEEE 1865-2014. TLM is often used in designs that includes multiple models, thus would greatly benefit from a more consistent structure to describe said models.

As a solution to lacking vendor neutrality and unintuitive methodology regarding the model description, a top-level *model* element is proposed. The *model* element replace everything specific to the model, and brings them under one element. One such model would be TLM. *Model* also packs a more structured methodology with a more consistent dependency pattern. With IP-XACT continuously being developed such feedback may provide very valuable to further development.

The future of IP-XACT is hard to predict precisely. It relies on the industry itself to get a stable foothold in the future industry. With IP-XACT being the only universal answer to a metadata centric design flow, one cannot help to think expansive in the years to come. To really fuel the expansion, it could use a more intuitive methodology and structure. Even though complexity and complicated structure can be solved by computer-aided designs, does not mean the designer nor the tool-vendor interpreting IP-XACT are machines too.

7 References

[1] P. J. Denning and T. G. Lewis, “Exponential Laws of Computing Growth | January 2017,” Communications of the ACM, January 2017. [Online]. Available: https://cacm.acm.org/magazines/2017/1/211094-exponential-laws-of-computing-growth/fulltext. [Accessed 3 March 2017].

[2] McKinsey & Company, “McK on Semiconductors Issue 3 2013,” Autumn 2013. [Online]. Available: http://www.mckinsey.com/client_service/semiconductors/~/media/32AE520663114C6EB1250BBDB92673C2.ashxBBDB92673C2.ashx&usg=AFQjCNGJm7QtPpIJCw5-WHf. [Accessed 11 June 2017].

[3] M. Groover and E. Zimmers, CAD/CAM: Computer-Aided Design and Manufacturing, Peason Education, 1983.

[4] D. C. Black, J. Donovan, B. Bunton and A. Keist, SystemC: From the Group Up, Austin, TX: Springer, 2010.

[5] B. Bailey, F. Balarin and M. McNamara, TLM-driven Design and Verification Methodology, Cadence Design Systems, Inc., 2010.

[6] Accellera, “IP-XACT Working Group,” [Online]. Available: http://accellera.org/activities/working-groups/ip-xact. [Accessed 18 June 2017].

[7] C. W. H. Strolenberg, “SoCs: Design tools -> Hard IP offers some hard-core values,” EETimes, 15 May 2000. [Online]. Available: https://www.design-reuse.com/articles/2552/socs-design-tools-hard-ip-offers-some-hard-core-values.html. [Accessed 8 June 2017].

[8] S. Vassiliadis, S. Wond and T. D. Hämäläinen, Embedded Computer Systems: Architectures, Modeling, and Simulation, Samos, Greece: Springer, 2006.

[9] Accellera, “SystemC,” Accellera, 3 November 2016. [Online]. Available: http://accellera.org/downloads/standards/systemc. [Accessed 8 June 2017].

[10] W3C, “Exstensible Markup Language (XML),” W3C, [Online]. Available: https://www.w3.org/XML/.

[11] W3C, “W3C,” [Online]. Available: https://www.w3.org/. [Accessed 2 June 2017].

[12] J. Surveyer, “Web Services Overview,” theopensourcery.com, [Online]. Available: http://theopensourcery.com/websover.htm. [Accessed 25 June 2017].

[13] IEEE-SA, “IEEE SA - 1685-2014 - IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within tool flows,” IEEE, [Online]. Available: https://standards.ieee.org/findstds/standard/1685-2014.html. [Accessed 13 February 2017].

[14] Accellera, “Home,” Accellera Systems Initiative, [Online]. Available: http://www.accellera.org/.

[15] IEEE, “IEEE-SA The IEEE Standards Association - Home,” IEEE, [Online]. Available: https://standards.ieee.org/.

[16] ARM, “Socrates Design Environment - ARM,” ARM, [Online]. Available: https://www.arm.com/products/system-ip/ip-tooling/socrates-design-environment.php.

[17] Magillem, “Magillem: EDA Front-end design and documentation software,” Magillem, [Online]. Available: http://www.magillem.com/.

[18] Xilinx, “Vivado_IP_Integrator_Backgrounder.pdf,” [Online]. Available: http://www.xilinx.com/publications/prod_mktg/vivado/Vivado_IP_Integrator_Backgrounder.pdf. [Accessed 13 February 2017].

[19] Duolos, “SystemC TLM-2.0,” Duolos, 2009. [Online]. Available: https://www.doulos.com/knowhow/systemc/tlm2/. [Accessed 8 June 2017].

[20] L. Fossati, “LEON_SystemC_FinalReport.pdf,” 2010. [Online]. Available: http://microelectronics.esa.int/core/ipdoc/LEON_SystemC_FinalReport.pdf. [Accessed 10 April 2017].

[21] I. SA, “1800-2012 - IEEE Standard for SystemVerilog--Unified Hardware Design, Specification, and Verification Language,” IEEE, 2012. [Online]. Available: https://standards.ieee.org/findstds/standard/1800-2012.html. [Accessed 8 June 2017].

[22] M. Simulink, “Simulink - Simulation and Model-Based Design,” MathWorks, 2017. [Online]. Available: https://se.mathworks.com/products/simulink.html. [Accessed 8 June 2017].

[23] Accellera, “UVM (Universal Verification Methodology),” Accellera, June 2014. [Online]. Available: http://accellera.org/downloads/standards/uvm. [Accessed 7 June 2017].

[24] Duolos, “Getting Started with OVM - Duolos,” Duolos, [Online]. Available: https://www.doulos.com/knowhow/sysverilog/ovm/tutorial_0/. [Accessed 8 June 2017].

[25] Accellera, “SystemC TLM,” Accellera, 2009. [Online]. Available: http://accellera.org/activities/working-groups/systemc-tlm/. [Accessed 8 June 2017].

[26] ASIC-WORLD, “SystemC Tools,” ASIC-WORLD, [Online]. Available: http://www.asic-world.com/systemc/tools.html. [Accessed 8 June 2017].

[27] Gaisler, “leon2-1.0.30-xst.pdf,” [Online]. Available: http://www.dit.upm.es/~str/ork/documents/leon2-1.0.30-xst.pdf. [Accessed 10 April 2017].

[28] Gaisler, “The SPARC Architecture Manual Version 8 - sparcv8.pdf,” [Online]. Available: http://gaisler.com/doc/sparcv8.pdf. [Accessed 04 April 2017].

[29] C. Gaisler, “Cobham Gaisler,” [Online]. Available: http://www.gaisler.com/. [Accessed 10 April 2017].

[30] GNU.org, “The GNU General Public Licence v3,” Free Software Foundation, Inc, 29 June 2007. [Online]. Available: https://www.gnu.org/licenses/gpl-3.0.en.html. [Accessed 8 June 2017].

[31] GNU.org, “GNU Lesser General Public License v3,” Free Software Foundation, Inc, 29 June 2007. [Online]. Available: https://www.gnu.org/licenses/lgpl-3.0.en.html. [Accessed 8 June 2017].

[32] Accellera, “IP-XACT,” Accellera, [Online]. Available: http://accellera.org/activities/working-groups/ip-xact. [Accessed 10 April 2017].

[33] J. Gaisler, “GitHub - Galland/LEON2: LEON2 SPARC CPU IP core LGPL by Gaisler Research,” Gaisler Research, 2013 April 9. [Online]. Available: https://github.com/Galland/LEON2. [Accessed 18 April 2017].

[34] W3C, “SOAP Version 1.2 Part 1: Messaging Framework (Second Edition),” W3C, 27 April 2017. [Online]. Available: https://www.w3.org/TR/soap12/. [Accessed 28 April 2017].

[35] ARM, “AMBA Open Specifications,” ARM, [Online]. Available: https://www.arm.com/products/amba-open-specifications.php. [Accessed 8 June 2017].

[36] Wikipedia, "IP-XACT - Wikipedia," Accellera, [Online]. Available: https://en.wikipedia.org/wiki/IP-XACT#Alphabetical_list_of_Companies.2FTools. [Accessed 5 June 2016].

[37] Tampere University of Technology, "Kaktus2," [Online]. Available: http://funbase.cs.tut.fi/. [Accessed 8 June 2017].

[38] EDAUtils, "Library of EDAUtils," [Online]. Available: http://www.edautils.com/. [Accessed 8 June 2017].

[39] Synopsis, "GenSys," [Online]. Available: https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/gensys.html. [Accessed 8 June 2017].

[40] DeFacto, "STAR - RTL IP INTEGRATION," [Online]. Available: http://www.defactotech.com/star-rtl-build-signoff/star-rtl-ip-integration. [Accessed 8 June 2017].

[41] Scineric, "Scrineric Workspace," [Online]. Available: scineric.csir.co.za. [Accessed 8 June 2017].

[42] AgniSys, "Cover All Verification with IP XACT," [Online]. Available: https://www.agnisys.com/products/idesignspec-uvm-register-generator/. [Accessed 8 June 2017].

[43] Xilinx, "Designing with Vivado IP Integrator," [Online]. Available: https://www.xilinx.com/video/hardware/designing-with-vivado-ip-integrator.html. [Accessed 8 June 2017].

[44] IEEE, "1801-2013 - IEEE Standard for Design and Verification of Low-Power Integrated Circuits," IEEE SA, 2015. [Online]. Available: https://standards.ieee.org/findstds/standard/1801-2013.html. [Accessed 7 June 2017].

[45] O. Kindgren, "Tales from Boyond the Register Map," 2 November 2016. [Online]. Available: http://olofkindgren.blogspot.no/2016/11/ip-xact-good-bad-and-outright-madness.html. [Accessed 18 June 2018].

[46] Accellera, "IP-XACT," [Online]. Available: http://accellera.org/downloads/standards/ip-xact. [Accessed 6 June 2017].

[47] B. Fuller, "EDA Tool Chain Too Complex," EE Times, 22 October 2013. [Online]. Available: http://www.eetimes.com/author.asp?doc_id=1319856. [Accessed 6 June 2017].

[48] GNU.org, "automake: Autotools Introduction," [Online]. Available: https://www.gnu.org/software/automake/manual/html_node/Autotools-Introduction.html. [Accessed 19 June 2017].

[49] Kitware, "CMake," [Online]. Available: https://cmake.org/. [Accessed 19 June 2017].

[50] E. D. Jack Donovan, "Design-reuse," Duolog Technologies, 3 March 2014. [Online]. Available: https://www.design-reuse.com/articles/34165/using-ip-xact-metadata-for-a-tlm-modeling-flow.html. [Accessed 2 June 2017].

[51] E. d. Kock, Interviewee, *Q & A with IP-XACT Working Group Chair.* [Interview]. May 2017.

[52] Wikipedia, "IP-XACT - Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/IP-XACT#Alphabetical_list_of_Companies.2FTools. [Accessed 03 June 2017].

[53] E. d. Kock, "Organization and Working Group Updates -- Video Presentations," Accellera, 11 November 2015. [Online]. Available: http://videos.accellera.org/updates.html#updatenov2015. [Accessed 18 April 2017].

[54] Tantawy, Guindi, Dessouky and Al-Imam, “Parameterized test patters methodology for laoyout design rule checking verification,” 24 March 2016. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/7440385/. [Accessed 5 June 2017].

[55] T. Wien, “B.3 Electronic Design Interchange Format (EDIF),” TU Wien, [Online]. Available: http://www.iue.tuwien.ac.at/phd/minixhofer/node53.html. [Accessed 05 June 2017].

[56] T. Wien, “EDIF - Wikipedia,” Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/EDIF#Evolution . [Accessed 5 June 2017].

[57] M. Slovick, “DAC 2016: Preparing for the Future of Electronic Design,” TTI, 16 June 2016. [Online]. Available: https://www.ttiinc.com/content/ttiinc/en/resources/marketeye/categories/passives/me-slovick-20160616.html. [Accessed 5 June 2017].

[58] Evatronix, “Straightforward IP Integration with IP-XACT RTL-TLM Switching,” Design-reuse.com, 3 June 2013. [Online]. Available: https://www.design-reuse.com/articles/18558/ip-xact-rtl-tlm-switching.html. [Accessed 27 April 2017].

8 Appendices

A. ApbSubSystem

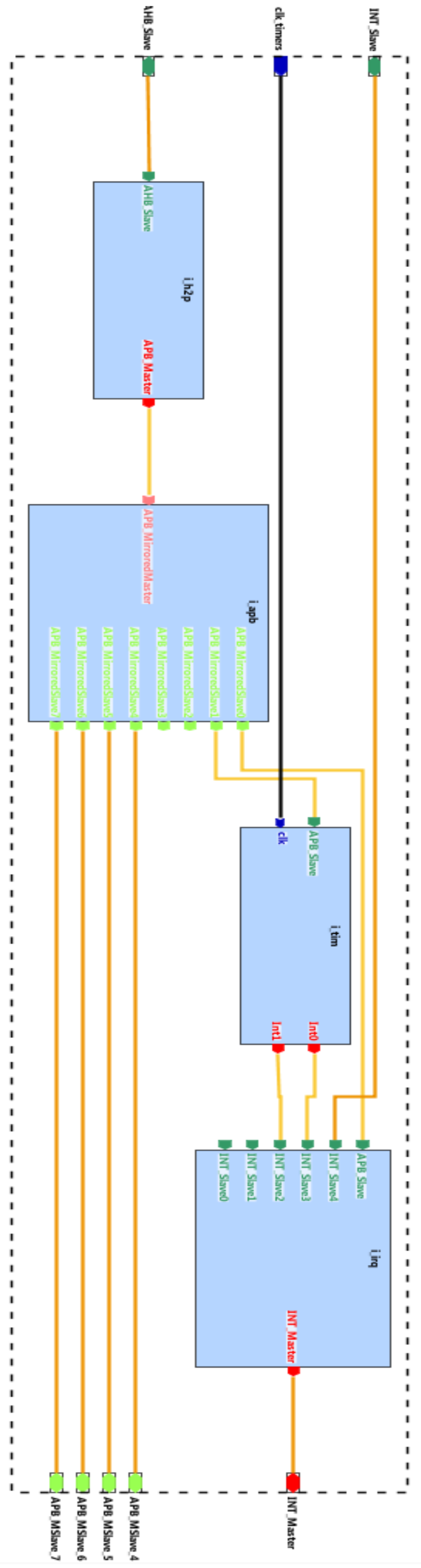


Figure 8-a ApbSubSystem without RNG module

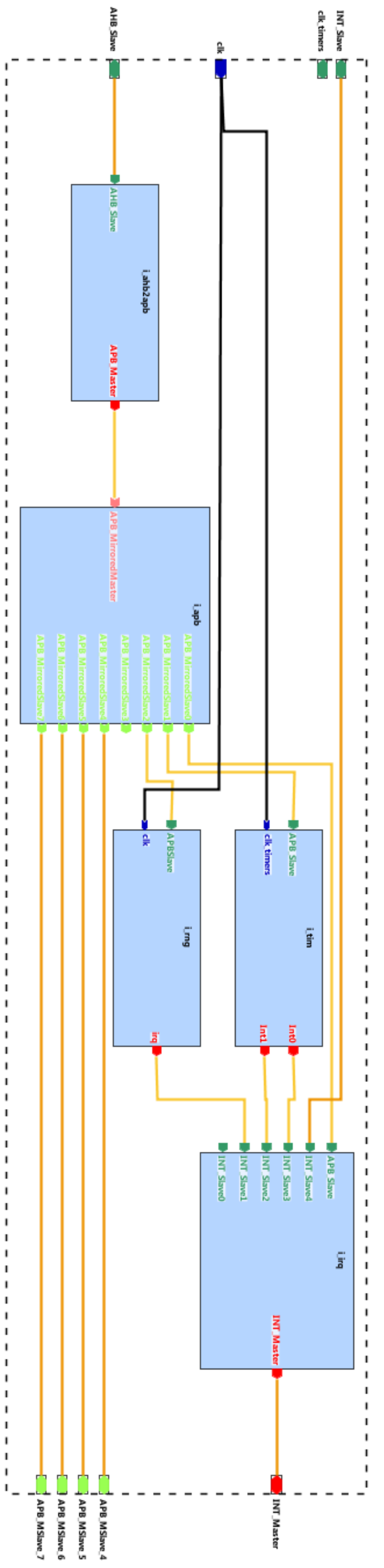


Figure 8-b ***ApbSubSystem with RNG module***

B. IP-XACT Description of RNG Module

```
<?xml version="1.0" encoding="UTF-8"?>
<spirit:component xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:spirit="http://www.spiritconsortium.org/XMLSchema/SPIRIT/1685-2009"
xsi:schemaLocation="http://www.spiritconsortium.org/XMLSchema/SPIRIT/1685-2009
http://www.spiritconsortium.org/XMLSchema/SPIRIT/1685-2009/index.xsd">
  <spirit:vendor>eirikpre</spirit:vendor>
  <spirit:library>tutorial</spirit:library>
  <spirit:name>rng</spirit:name>
  <spirit:version>1.0</spirit:version>
  <spirit:busInterfaces>
    <spirit:busInterface>
      <spirit:name>APBSlave</spirit:name>
      <spirit:busType spirit:library="AMBA2" spirit:name="APB"
spirit:vendor="amba.com" spirit:version="r2p0_4"/>
      <spirit:abstractionType spirit:library="abstractiondef.tlm"
spirit:name="apb_pv" spirit:vendor="spiritconsortium.org"
spirit:version="1.4"/>
      <spirit:slave>
        <spirit:memoryMapRef spirit:memoryMapRef="APB"/>
      </spirit:slave>
      <spirit:portMaps>
        <spirit:portMap>
          <spirit:logicalPort>
            <spirit:name>PV_TRANS</spirit:name>
          </spirit:logicalPort>
          <spirit:physicalPort>
            <spirit:name>apb_slave_port</spirit:name>
          </spirit:physicalPort>
        </spirit:portMap>
      </spirit:portMaps>
      <spirit:bitsInLau>8</spirit:bitsInLau>
    </spirit:busInterface>
    <spirit:busInterface>
      <spirit:name>irq</spirit:name>
      <spirit:busType spirit:library="busdef.interrupt"
spirit:name="interrupt" spirit:vendor="spiritconsortium.org"
spirit:version="1.0"/>
      <spirit:abstractionType spirit:library="busdef.interrupt"
spirit:name="interrupt_rtl" spirit:vendor="spiritconsortium.org"
spirit:version="1.0"/>
      <spirit:master/>
      <spirit:portMaps>
        <spirit:portMap>
          <spirit:logicalPort>
            <spirit:name>IRQ</spirit:name>
          </spirit:logicalPort>
          <spirit:physicalPort>
            <spirit:name>irq</spirit:name>
          </spirit:physicalPort>
        </spirit:portMap>
      </spirit:portMaps>
      <spirit:bitsInLau>8</spirit:bitsInLau>
    </spirit:busInterface>
  </spirit:busInterfaces>
  <spirit:memoryMaps>
    <spirit:memoryMap>
      <spirit:name>APB</spirit:name>
      <spirit:addressBlock>
        <spirit:name>RNG</spirit:name>
        <spirit:baseAddress>0x0</spirit:baseAddress>
        <spirit:range>4096</spirit:range>
        <spirit:width>32</spirit:width>
        <spirit:register>
          <spirit:name>CONFIG</spirit:name>
          <spirit:addressOffset>0x504</spirit:addressOffset>
          <spirit:size>32</spirit:size>
          <spirit:access>read-write</spirit:access>
        </spirit:register>
        <spirit:register>
          <spirit:name>VALUE</spirit:name>
          <spirit:addressOffset>0x508</spirit:addressOffset>
          <spirit:size>32</spirit:size>
          <spirit:access>read-only</spirit:access>
          <spirit:field>
            <spirit:name>VALUE</spirit:name>
            <spirit:bitOffset>0</spirit:bitOffset>
            <spirit:bitWidth>8</spirit:bitWidth>
          </spirit:field>
          <spirit:field>
            <spirit:name>DERCEN</spirit:name>
            <spirit:bitOffset>8</spirit:bitOffset>
            <spirit:bitWidth>1</spirit:bitWidth>
          </spirit:field>
        </spirit:register>
        <spirit:register>
          <spirit:name>POWER</spirit:name>
          <spirit:addressOffset>0xFFC</spirit:addressOffset>
          <spirit:size>32</spirit:size>
          <spirit:access>read-write</spirit:access>
          <spirit:field>
            <spirit:name>POWER</spirit:name>
            <spirit:description>    </spirit:description>
            <spirit:bitOffset>0</spirit:bitOffset>
            <spirit:bitWidth>1</spirit:bitWidth>
          </spirit:field>
        </spirit:register>
        <spirit:register>
          <spirit:name>START</spirit:name>
          <spirit:addressOffset>0x400</spirit:addressOffset>
          <spirit:size>32</spirit:size>
          <spirit:access>write-only</spirit:access>
          <spirit:field>
            <spirit:name>START</spirit:name>
            <spirit:bitOffset>0</spirit:bitOffset>
            <spirit:bitWidth>1</spirit:bitWidth>
          </spirit:field>
        </spirit:register>
      </spirit:addressBlock>
    </spirit:memoryMap>
  </spirit:memoryMaps>
  <spirit:model>
    <spirit:views>
      <spirit:view>
        <spirit:name>TLM_PV</spirit:name>
        <spirit:envIdentifier>:*Simulation:</spirit:envIdentifier>
        <spirit:language>SystemC</spirit:language>
        <spirit:modelName>rng</spirit:modelName>
        <spirit:fileSetRef>
          <spirit:localName>systemC</spirit:localName>
        </spirit:fileSetRef>
      </spirit:view>
      <spirit:view>
        <spirit:name>RTL</spirit:name>
<spirit:envIdentifier>EnvId:EnvId:EnvId</spirit:envIdentifier>
        <spirit:language>SystemVerilog</spirit:language>
        <spirit:modelName>rng</spirit:modelName>
      </spirit:view>
    </spirit:views>
    <spirit:ports>
      <spirit:port>
        <spirit:name>apb_slave_port</spirit:name>
        <spirit:transactional>
```

```
          <spirit:service>
            <spirit:initiative>provides</spirit:initiative>
            <spirit:serviceTypeDefs>
              <spirit:serviceTypeDef>
                <spirit:typeName
spirit:implicit="false">OSCI_TLM_PV</spirit:typeName>
                <spirit:parameters>
                  <spirit:parameter>
                    <spirit:name>typedef1</spirit:name>
                    <spirit:value>ADDRESS_TYPE</spirit:value>
                  </spirit:parameter>
                  <spirit:parameter>
                    <spirit:name>typedef2</spirit:name>
                    <spirit:value>DATA_TYPE</spirit:value>
                  </spirit:parameter>
                </spirit:parameters>
              </spirit:serviceTypeDef>
            </spirit:serviceTypeDefs>
          </spirit:service>
        </spirit:transactional>
      </spirit:port>
      <spirit:port>
        <spirit:name>irq</spirit:name>
        <spirit:wire>
          <spirit:direction>out</spirit:direction>
          <spirit:vector>
            <spirit:left>0</spirit:left>
            <spirit:right>0</spirit:right>
          </spirit:vector>
          <spirit:wireTypeDefs>
            <spirit:wireTypeDef>
              <spirit:typeName>int</spirit:typeName>
              <spirit:viewNameRef>TLM_PV</spirit:viewNameRef>
            </spirit:wireTypeDef>
          </spirit:wireTypeDefs>
        </spirit:wire>
      </spirit:port>
      <spirit:port>
        <spirit:name>clk</spirit:name>
        <spirit:wire>
          <spirit:direction>in</spirit:direction>
          <spirit:wireTypeDefs>
            <spirit:wireTypeDef>
              <spirit:typeName>int</spirit:typeName>
              <spirit:viewNameRef>TLM_PV</spirit:viewNameRef>
            </spirit:wireTypeDef>
          </spirit:wireTypeDefs>
        </spirit:wire>
      </spirit:port>
    </spirit:ports>
  </spirit:model>
  <spirit:fileSets>
    <spirit:fileSet>
      <spirit:name>systemC</spirit:name>
      <spirit:file>
        <spirit:name>rng.h</spirit:name>
        <spirit:fileType>systemCSource</spirit:fileType>
        <spirit:isIncludeFile
spirit:externalDeclarations="true">true</spirit:isIncludeFile>
        <spirit:dependency>../../PV</spirit:dependency>
      </spirit:file>
    </spirit:fileSet>
  </spirit:fileSets>
</spirit:component>
```