Universidade Federal do Rio Grande do Sul
Instituto de Informática

Graduation Project

Performed at STMicroelectronics

Automatic Generation of Register Side-Effect Test in Embedded Software

Arthur KALSING

Computer Engineering

July 2014

STMicroelectronics
12, Rue Jules HOROWITZ
38019 Grenoble Cedex FRANCE

Industrial Advisor
Laurent BERNARD
laurent-andre.bernard@st.com

University Supervisor
Renato Perez Ribas
rpribas@inf.ufrgs.br

Acknowledgements

I am grateful INPG-Ensimag for providing this unique opportunity of working on this great project in a foreign country. Remarkably Laurent Bernard, my Industrial Advisor, for the project proposal, Professor Matthieu Moy, my Supervisor, and Professor Florence Maraninchi, Course Manager, for having accepted this proposal. I also thank Professor Renato Perez Ribas for helping me publish this work in my home university – UFRGS. This end of studies project is a great experience for any interchange student and will surely add value for my future as engineer.

I would like to express my gratitude to all System Platforms Group members that worked with me so far. This experience was remarkably important for my scientific, technique, and personal development. For sure that was my best work experience and I'm gladly to have worked with such coworkers. I have much and more to say about this amazing team, but let's synthesize:

First of all to Laurent Bernard, who supervised me over the last six months, from the definition of the subject until the revision of my report, always getting time to teach me and check my work (even on vacancies and weekends!). For his patience trying to understand my "*Frenguese*" (French with Portuguese accent) and his motivational way of being what surely helped me maintain the focus and patience during the project.

To Kamlesh Pathak for his support over *spirit2regtest* and IP-XACT descriptions, for the important time spent in meeting when he came to Grenoble, and for being regardful with my work even at India. I also thank Jerome Cornet and Mohamed Korbosli for their support over *tlm_register_bank* and implementation ideas.

To Laurent Maillet-Contoz and Antoine Perrin for showing me the important paper of the manager that must lead the team raising the moral and encouraging its members to give their best.

To Ayoub Arfaoui and Aurelien Chichignoud my student mates, for their friendship, for teaching me how to be more confident and less stressed, and for the talks beyond number about Islam, French and Brazilian culture.

To all SPG members, especially Herve Broquin, my box neighbor, for the relaxed work environment, with their jests and laughter, animated breakfasts, coffee breaks and barbecues, showing me that even guys on suit and tie can have fun at work.

Finally I would like to thank my family for encouraging me, showing their pride and faith on me. Thank you all.

Abstract

The techniques and procedures for System-on-Chips (SoC) design are constantly being improved, not only because of the fast evolution of this area (as predicted by Moore's Law) but mainly due to the high complexity that such task acquires. Specialized R&D teams in semiconductors companies work in the improvement of this process, researching better methodologies and technologies in order to gain market advantage.

This graduation project, developed within System Platforms Group (SPG) at STMicroelectronics, aims at improving one of the steps of the company's tests flow, more specifically the register bank test in transactional level of semiconductor's conception. Precisely, this project aims at improving a tool which will help the user to set up a complete register bank test, including test for special register behaviors, automatically generating the embedded code to be executed.

Through this text is presented the work performed over the project step-by-step, from the analysis of the existent methodology to the tests in real industry IPs using the improved process.

Resumo

As técnicas e processos de concepção de Systems-on-Chip (SoC) estão em constante evolução, não apenas devido a rápida evolução deste ramo da indústria (como previsto pela lei de Moore) mas principalmente devido a alta complexidade que esta tarefa vem adquirindo. Equipes especializadas em P&D trabalham nas empresas de semicondutores pesquisando novas metodologias e tecnologias para alcançar vantagem comercial.

Este trabalho de graduação, desenvolvido dentro do Grupo de Plataformas e Sistemas da STMicroelectronics, visa o melhoramento de um dos processos de validação de design da companhia, mais especificamente o teste do banco de registradores em nível transacional. Precisamente, este projeto objetiva o aprimoramento de uma ferramenta de teste que auxilia o usuário na criação de uma rotina de testes completa para bancos de registradores, incluindo testes para registradores com comportamentos específicos, automaticamente gerando o código embarcado para ser executado na plataforma.

Através deste trabalho será apresentado todo o desenvolvimento do projeto, desde a análise da metodologia existente na companhia até os testes em plataformas industriais utilizando software gerado pela ferramenta aprimorada.

Keywords

Transactional Level Modeling, TLM Validation, IP-XACT, Register Bank Test, Side-Effects, System on Chip Design, Embedded Software.

Table of Contents

Table of Figures

Glossary

- **EDA (ECAD):** Electronic Design Automation, category of software used for electronic systems development.
- **HDL:** Hardware Description Language, formal language for electronic and digital systems description.
- **IEEE:** Institute of Electrical and Electronics Engineers, world's greatest association of engineers, responsible for most of technological innovation and standards.
- **IP:** Intellectual Property, term commonly used to refer a digital component with industrial design rights.
- **IP-XACT:** Formal description language based in XML for IPs description.
- **ISS:** Instruction Set Simulator, virtual simulation of a processor with a defined instruction set.
- **NoC:** Network on Chip, term commonly used to refer complex digital systems which require more than the usual communication systems.
- **RTL:** Register Transfer Level, design abstraction for digital circuits, uses HDL based languages, more close to the circuit itself.
- **SoC:** System on Chip, another term commonly used to refer complex digital systems, a circuit with several digital components.
- **SPG:** Systems Platform Group, STMicroelectronics division in charge of R&D over virtual platforms and interfaces.
- **ST:** Abbreviation for STMicroelectronics.
- **SystemC:** C++ library and simulation Kernel, dedicated to modeling circuits and SoCs.
- **TLM:** Transactional Level Modeling, high-level design abstraction for modeling complex digital systems.
- **XML:** EXtensible Markup Language, well-formed human-readable and machine-readable description language.

1. Introduction

To achieve fast and reliable design of systems on chip (SoCs) a complete and incremental flow of conception is used today inside semiconductors industry. Each stage of the flow validates a specific part of the system attempting to avoid, at maximum, the massive simulation needed by the lowest levels of conception. One of the highest levels of the conception flow is the Transactional Level Modeling (TLM) which holds mainly the functional test of the platform.

In TLM the electric behavior of the components is abstracted focusing the simulation in the behavior and communication between the platform IPs. The IPs are modeled with few basic components as communication ports, registers and processes, only what is strictly needed for the functional simulation. These factors made the TLM simulation far lighter comparing with other low-level simulations, and can validate most of the characteristics that would be validated after. This raises the productivity since it provides a simulation platform for software testing before having a silicon prototype of the SoC, in other words, it allows parallel software and hardware and reduces time to market.

One of the most important tests to be performed over TLM platform is the register bank test. This validation includes the verification of initial value, access rights and side-effects behavior of the registers. Over the last decades the components have been increasing more and more their number of registers, turning the register test task into a precise and repetitive job, normally resulting in a lot of mistakes.

In order to automate the register bank test, a tool was created to automatically generate embedded software for the test. This tool follows the flow of TLM simulation beginning with the IP-XACT description of the component, where the register bank is formally described, and generating a quite complete register test. However, this tool still has several limitations, especially with IPs that have unusual registers behaviors. This graduation project will focus in the enhancement of this tool, improving it in order to enable the easy generation of a much more complete register test and management of side-effects.

1.1 The Semiconductors Market

The ability to transform energy into intelligence changes our lives day-by-day, whether by our communication, transport, entertainment, healthcare, or any other feature of our lives. Digital components are everywhere: computers, cellphones, tablets, digital cameras, in our houses, in our cars, more and more used and explored each day. Clearly, semiconductors became essential for human development.

Consequently the semiconductors market has the most significant growth of all industrial sectors since its emergence. Early predicted by the famous Moore's law the number of transistors on silicon chips doubles every eighteen months, providing more powerful and less costly electronic devices. To point some numbers, even with the recent European crisis and the weak recovery of the North American economy, the worldwide semiconductor has revenue of US$299 billion in 2012[1].

1.2 High level SoC design

As the name suggests, systems-on-chip are complete systems composed of both hardware and software, embedded in a single silicon chip. In other words, we can have a computer with all its components in a single chip. They are normally composed of a core subsystem surrounded by digital, analog, mixed-signal and any sort of needed IPs. Such architecture points the advantage of using SoCs: they are especially designed for its finality being more efficient, cheap and reliable. However, such completeness is achieved only after meticulous design and manufacture due to their complexity and bug high costs.

System-on-chip design process is based in the description of the architecture in several models called abstraction levels. Each model is characterized by part of the system in terms of details expressiveness, allowing the verification of the system in several stages. The first stages are used to verify the functional characteristics of the SoC taking advantage of simpler abstraction levels to perform fast simulation. Once the design flow reaches the latest levels the system is already verified for most of its characteristics and only more precise characteristics must be certificated, as the circuit timing for example. In this manner, the engineers successfully encounter a way to avoid time costly simulations and easy recuperate from bugs found earlier in the process.

However, the hardware of a SoC needs its corresponding embedded software. Formerly the software was the last thing to be developed since the physical chip was needed to start the development. To solve this problem an important new concept entered the design flow in the last decade, the transitional level modeling (TLM). Thanks to the apparition of

[1]Source: Gartner, see bibliography for links.

TLM, now the embedded software can be developed in parallel with the hardware, reducing much more the time to market. The diagram below (Figure 1 - SoC development abstraction levels) shows the current SoC development flow.

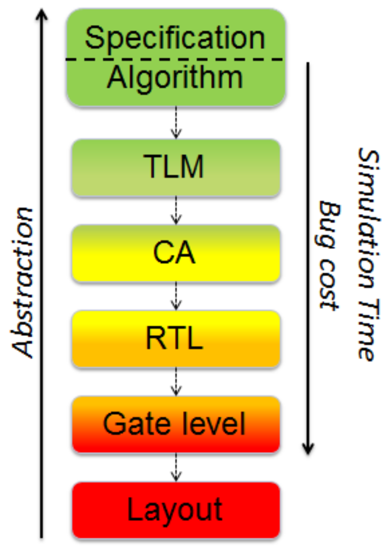


Figure 1 – SoC development abstraction levels

There are six main levels of abstraction used in the design flow today. For now we will focus in the first two levels of abstraction where most of the work of this project was performed. The lower levels would be naturally affected as can be seen in the diagram, but since they are not our focus we will skip them from this project. Better explanation of these abstraction levels is given in the background chapter.

1.3 IP Register Bank Verification

Register bank test is an essential step of IP verification process due to the increasingly use of registers in complex SoC. This test must be the more precise and complete possible ensuring correct initialization and operation for registers. Imagine that a single inverse bit in a configuration register initial value can completely disable a component module. In a worst-case scenario, a register fault may advance to the next level of design until the layout level thereby causing losses of millions of dollars.

This test is based on sequences of reads and writes to the registers, depending on their characteristics. Normally it begins by assuring the right initial value of the register through a read of the register value and a comparison with the specification value. Next, the access mode of the register is tested, verifying if it is possible to read and write according to the register access rights. Finally, if the register owns some advanced behavior this must be also tested to ensure correct operation. These three tests, apparently simple, can be very complex depending on the architecture of the register bank and particularly due to the behavior of some registers which can cause side-effects.

The tests are normally executed by especially designed embedded software (eSW), over TLM and RTL platforms. The advantage in this case is that we can advance the test software in TLM level, validate it, and after use the same code to test the component in RTL level. At first, functional verification is performed over TLM assuring the component right operation. After, on RTL, a more exhaustive verification is performed checking also timings and manufacture issues as stuck-at bits. The exhaustiveness of the test depends on the level of verification desired, in some cases each bit of each register must be exercised independently, resulting in time costly tests.

1.3.1 Automatically generated eSW

Standardization of descriptions has much contributed for register tests, thanks to that most of the embedded software for register test can be automatically generated. The last IP-XACT schema release is capable of giving most of the registers bank architecture which, in turn, is translated by a converting tool into eSW for register test.

However, to maintain the standard consistent only the most general and basic characteristics are presented in the description. For IPs with more complex registers constructions, as signaling registers for example, the standard description is insufficiently informative, preventing the complete automation of the test. In other words, most of the register test process can be automated leaving only advanced and specific characteristics to be manually implemented.

1.3.2 Known Limitations

Advanced characteristics for registers are the main concerns in register test today. As explained before, initial value is one of the basic tests to ensure correctness. Now imagine a register bank where an arbitrary register can change one bit of the next one, how can we ensure the initial value of the affected register? Complex cases can be achieved as multiple bounded registers, or a register that changes its value on read operation for example. Anyways, each one must be tested.

Besides, the automatic generation of embedded software for tests today relies on IP-XACT standards, what means that advanced characteristics are not covered by the schemas. Consequently, the generated code cannot prevent tests neither errors from advanced behaviors, in this case the user must develop the tests by itself.

1.4 Work Objectives

Aiming the improvement of register test generation and knowing the limitations of the current flow, some general objectives were defined for this project:

- **Analyze current register test flow**, in order to identify problems and difficulties faced by users today, identify missing features that can be implemented.
- **Upgrade register test generation**, by improving the eSW generator tool and possibly the methodology solving as much as possible the issues encountered in the analysis stage.
- **Evaluate the results**, by testing the new solution over real industry IPs and deploying the test software tool for users.

1.5 Paper Structure

In the remainder of this text we gradually present the work done over this project, starting with the required background (chapter 2) to understanding exactly the environment in which we are inserted. Chapter 3 describes the analysis of the work done before to automatize the test process including the methodology used, finishing with a set of upgrade specifications for the improved tool. In the following chapter (Chapter 4) we describe the implementation of the upgrade divided in two stages. The last chapter presents how we validated the new test process by setting up real test platforms and performing a user's survey.

2. Background

The analysis of the test process was an important step for this project, through which was possible to understand the automated flow of conception of tests for registers bank. With this purpose, training sessions, meetings and documentation analysis were done over the first months of the project.

As briefly explained in the previous chapter, the registers bank test is executed over TLM platform, programming a virtual processor with embedded software to exercise each register of the IP. This basically means that two elements are needed to perform the test, the virtual platform with the IP transactional model and the embedded software for testing.

The TLM platform consists of a simulated core bounded to the IP aimed for the test. Other components can be also included if the operation of the IP depends on them. However, the focus of the tests is on the IP, probably the only component under design while the others are normally reused from the company database or provided by customers. The core can be a generic native CPU performing host code execution (HCE) or an instruction set simulator (ISS) that really simulates the processor. Finally, the IP is obtained using a converter tool that outputs the IP model skeleton by parsing the IP-XACT description, in our case the tool is called *tlm_skeleton* and was developed by ST engineers.

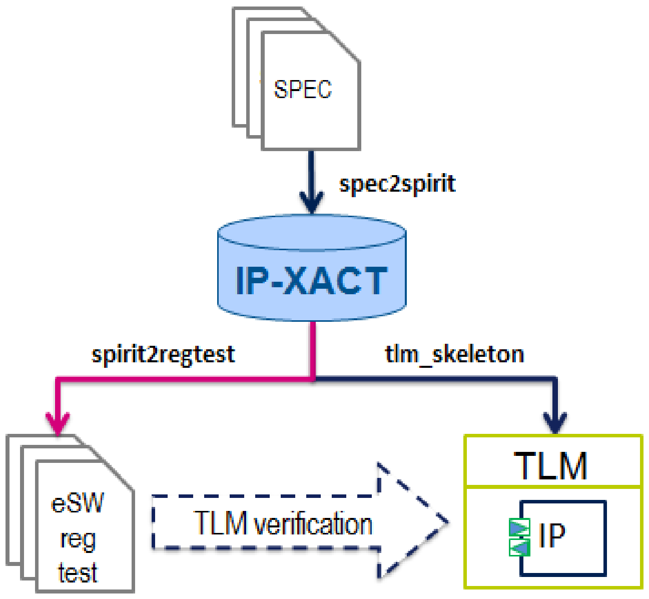


Figure 2 – Register Bank test flow

Once the TLM platform is operational it is necessary to generate the embedded software (eSW) for register test. Also, using the IP-XACT description, a converter tool called *spirit2regtest* converts the register bank description into a chain of tests in eSW which are loaded into the platform executing core. Finally, with the platform setup and the eSW loaded, the verification can be performed. The flux diagram above (Figure 2) shows the steps needed to perform the verification.

This flow goes on until it reaches the physical prototype of the SoC. An important remark is that the IP-XACT description can also be automatically generated from a well-defined specification document using *spec2spirit*. In a completely automated design flow the specification is meant to be the first and unique interface with the human. In the next chapters each conception stage will be explored with emphasis on register bank and register test characteristics.

2.1 Register test flow analysis

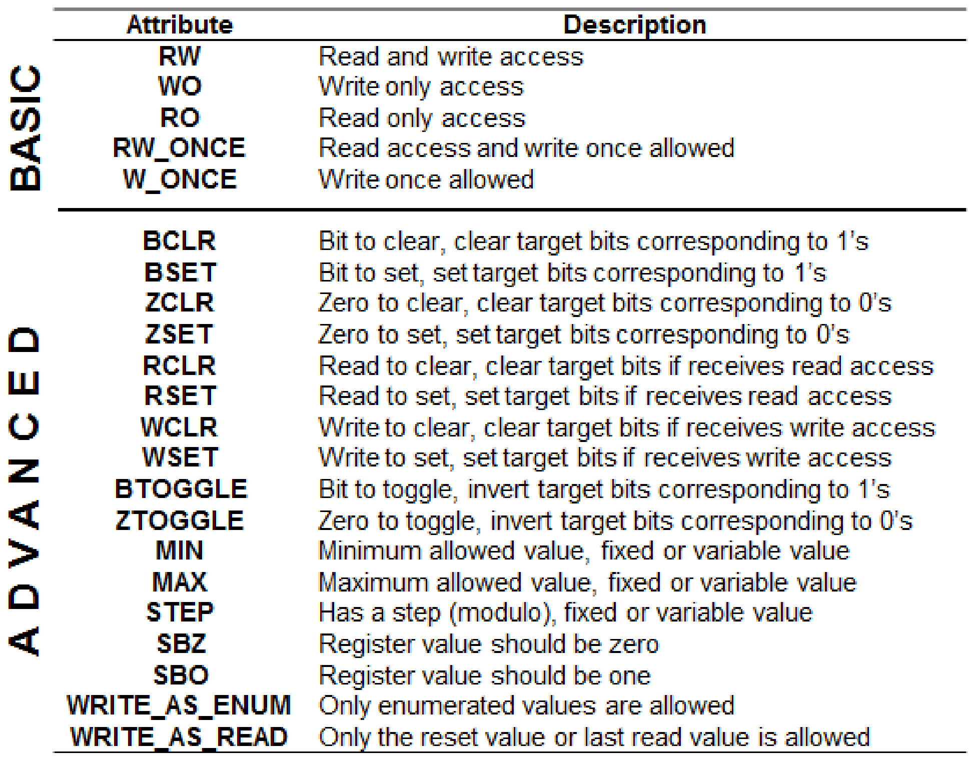
Registers are essential components for digital devices. They provide a way to get information into and out the component through memory address reads and writes. They can be used as memory for I/O, communication channel buffers, also in more architectural ways as control and status registers. The register interface often occupies the majority of the IP's manual and may be complex in a variety of dimensions.

Though this might seem quite simple, register and their fields' characteristics are the main concern of this project. Ensure the correct operation of the registers is essential to ensure the correct operation of the IP since the last one directly relies on registers information. Even after decades developing digital components, there is still no main standard for register characteristics and virtually everything is possible. However, some characteristics are more commonly used in chip design, for this work we decided to divide register characteristics in the following two sets:

Basic Attributes: Commonly called access mode, the basic attribute is mandatory information since it controls the register basic operation.

Advanced Attributes: Besides the access mode several registers fields have also special characteristics which can trigger side-effects on themselves or on another registers fields. Some of these characteristics are more commonly used, here called Advanced Attributes.

The whole considered set of attributes is illustrated in the table below (Figure 3).



	Attribute	Description
BASIC	RW	Read and write access
	WO	Write only access
	RO	Read only access
	RW_ONCE	Read access and write once allowed
	W_ONCE	Write once allowed
ADVANCED	BCLR	Bit to clear, clear target bits corresponding to 1's
	BSET	Bit to set, set target bits corresponding to 1's
	ZCLR	Zero to clear, clear target bits corresponding to 0's
	ZSET	Zero to set, set target bits corresponding to 0's
	RCLR	Read to clear, clear target bits if receives read access
	RSET	Read to set, set target bits if receives read access
	WCLR	Write to clear, clear target bits if receives write access
	WSET	Write to set, set target bits if receives write access
	BTOGGLE	Bit to toggle, invert target bits corresponding to 1's
	ZTOGGLE	Zero to toggle, invert target bits corresponding to 0's
	MIN	Minimum allowed value, fixed or variable value
	MAX	Maximum allowed value, fixed or variable value
	STEP	Has a step (modulo), fixed or variable value
	SBZ	Register value should be zero
	SBO	Register value should be one
	WRITE_AS_ENUM	Only enumerated values are allowed
	WRITE_AS_READ	Only the reset value or last read value is allowed

Figure 3 – Registers characteristics

2.2 *Specification Level*

This is the design flow first level of abstraction. In this level the architects describe the IP or the system with all needed components, protocols, specifications and functional details. Low-level details as timing and power consumption are explored later. The product of this level as the name suggests is an initial specification, provided normally on datasheet style, using pre-formatted tables in a document with the description. Today is also possible the use frameworks to help specifying, taking advantage of components database for reuse for example. Once the description is finished we begin the automated design flow by converting this document in a more formal and machine-readable format, as IP-XACT.

2.2.1 IP-XACT

Based on e**X**tensible **M**arkup **L**anguage (XML) format, IP-XACT is a formal hardware description standard created to improve semiconductors design process. HDL languages were revolutionary concepts in the semiconductors world allowing synthesis and accurate simulation of hardware, even so their complexity in terms of description hamper high-level approaches as reuse and automation. As an endeavor to avoid complicated descriptions and to provide a global standard, IP-XACT was created to be a mid-term hardware description between the initial specification and the HDL (Hardware Description Language).

IP-XACT is an innovative technology and has much improved the productivity of semiconductors design flow in the industry. Four main goals are commonly pointed as its objectives:

- Ensure delivery of compatible component descriptions between vendors.
- Enable exchanging of complex component libraries between electronic design automation (EDA) tools.
- Describe configurable components using metadata.
- Enable the provision of EDA vendor-neutral tools.

Basically, provide an easier way to describe, exchange and process automation following a global standard.

There are two main interesting characteristics behind the use of IP-XACT. First, the use of XML as description language, this both human-readable and machine-readable format allows easy parsing and conversion of the design description to models and tests between other abstraction levels. The language is well settled and a vast set of APIs have been already developed to aid developers processing XML data. The second main characteristic is the creation of a global industry standard providing a common format to describe IPs, their interfaces, memory mapping, registers bank, bindings, net-list, among others. This standardization ensures coherency between IPs from different vendors, improving integration, reuse and especially automation for the design flow. IP-XACT can be seen as a great database for all companies, as in the Figure 4 where we can insert and abstract data on different formats.

This technology is being improved since its first schema release in 2004, when it was still called SPIRIT. Accellera® has set up a committee called SPIRIT Consortium, where most of the great semiconductors companies have contributors, and finally in 2009 the first IEEE approved version was released. This is the starting point of this project, more specifically the approved IP-XACT IEEE 1685-2009, when the first traces of registers side-effects have been added to the standard.

Nowadays EDA companies have already developed a strong ecosystem around IP-XACT with converters, CAD tools and frameworks. Designers are free to choose which alternative better suits or so develop their own tools. STMicroelectronics has a complete set of tools for IP-XACT which can automatically create and set up most of the virtual platform, leaving only the specific behavior of the IPs to be coded by the users.

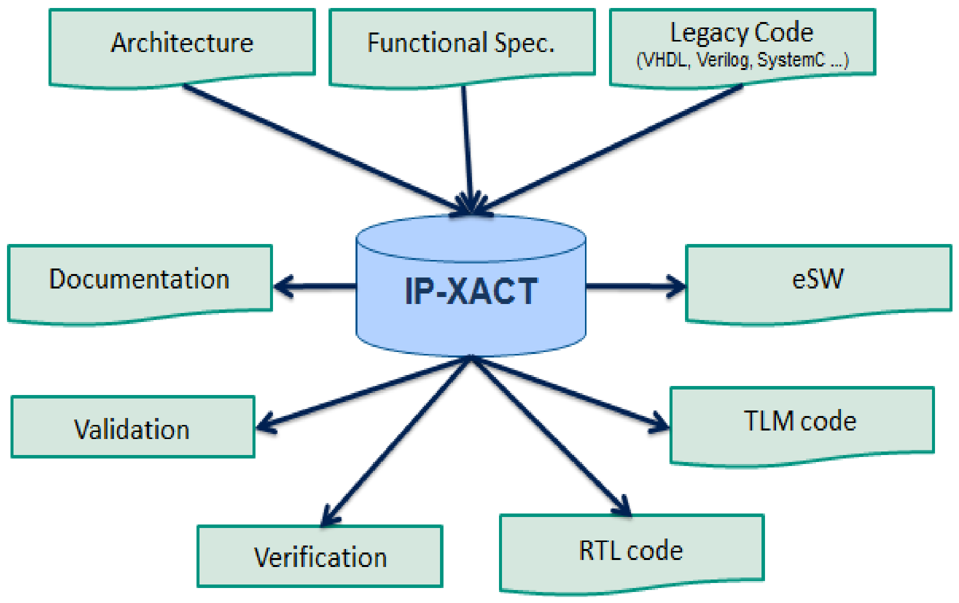


Figure 4 – IP-XACT ecosystem

Once we have the IP-XACT description one can proceed to the next stages of development of the SoC. In the design flow we can consider this document as the output of the specification level and the input for some of the next levels, as TLM and RTL, for example.

2.2.2 Registers bank description on IP-XACT

Register description on IP-XACT underwent certain changes through his official releases. The first traces of special behavior on register began to appear in the last IEEE approved version, what could mean a possible standardization for some attributes. However, key information is still missing in the schema, what prevents the complete automation of the test regarding advanced attributes. Even so, the schema clearly illustrates that it is possible to add information using vendor extensions.

The IP register bank is declared inside the component memory map structure, more specifically inside an *addressBlock*. Once an *addressBlock* is declared, one can start adding the registers by blocks (*registerFile*) or individually (*register*). The *register* structure is very well detailed with a complete set of information for register, allowing also the declaration of bit fields (*field*). Finally, within the *field* structure we can find the leaves of advanced attributes. An example of IP-XACT description for register is given below.

```
<spirit:register>
  <spirit:name>MYIP_REG1</spirit:name>
  <spirit:description>Read-write 8 bits register
  </spirit:description>
  <spirit:addressOffset>0x003B</spirit:addressOffset>
  <spirit:size>8</spirit:size>
  <spirit:access>read-write</spirit:access>
  <spirit:reset>
    <spirit:value>0xFF</spirit:value>
  </spirit:reset>
  <spirit:field>
    <spirit:name>ZSET</spirit:name>
    <spirit:description>Zero set field </spirit:description>
    <spirit:bitOffset>4</spirit:bitOffset>
    <spirit:bitWidth>2</spirit:bitWidth>
    <spirit:access>read-write</spirit:access>
  </spirit:field>
</spirit:register>
```

Regarding the possible attributes in relation with the two defined groups, IP-XACT covers all the basic attributes and part of the advanced attributes. The *access* element allows the declaration of the access mode defined as: read-write, read-only, write-only, read-writeOnce or writeOnce. *EnumeratedValues, modifiedWriteValue, writeValueConstraint and readAction* allow the declaration of the whole set of advanced attributes, however, the schema does not mention any target, this means that these attributes are self-applied to the fields. The diagram below (Figure 5) illustrates the *register* and *field* structures and its attributes.

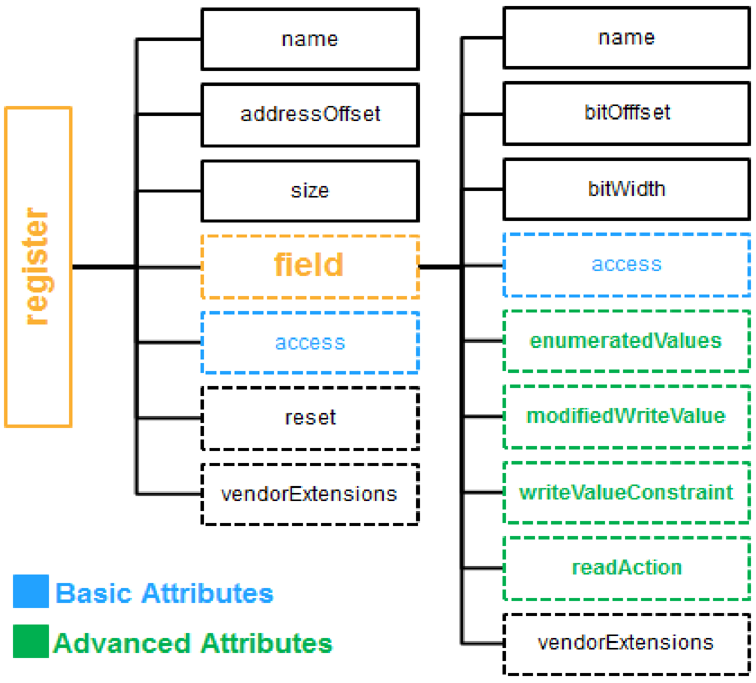


Figure 5 – Register IP-XACT structure

2.3 *Transaction Level Modeling*

Transaction Level Modeling is a high-level approach for digital systems design and simulation, register-accurate and with bit true-behavior, focused on the component functionality. This means that only the strictly needed elements of the component are modeled as data transfer, interrupts and memory mapping. Real implementation characteristics as the communication protocol, timings and power consumption are normally abstracted providing more flexibility and productivity on building virtual platforms for functional validation and verification, embedded software development and architecture exploration. Since its creation TLM has become the industry standard for creating inter operable transaction-level platforms.

This abstraction level was created to solve some critical problems of the design flow as the delayed embedded software development (incompatible with the time to market), increased simulation time of lower levels, and also to provide a way to validate the system before the synthesis solving bugs at design time. The diagram below (Figure 6) shows a detailed view of each level with their objectives and characteristics.

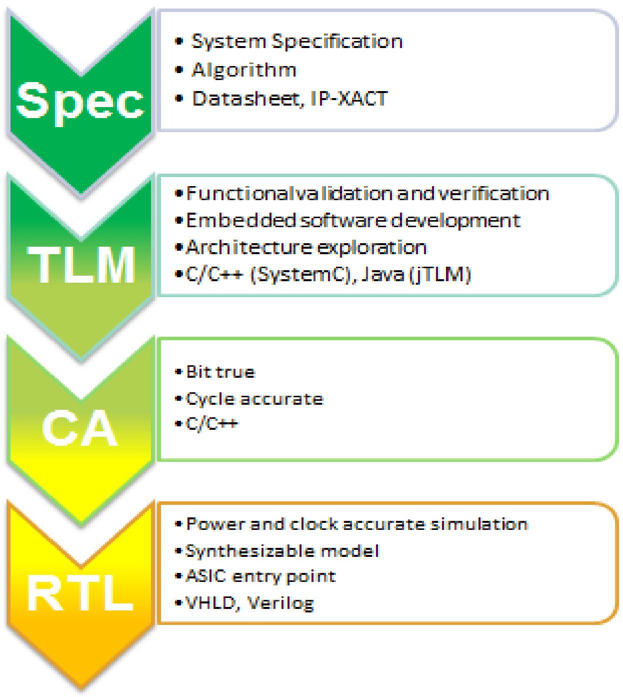

Spec
• System Specification
• Algorithm
• Datasheet, IP-XACT
TLM
• Functional validation and verification
• Embedded software development
• Architecture exploration
• C/C++ (SystemC), Java (jTLM)
CA
• Bit true
• Cycle accurate
• C/C++
RTL
• Power and clock accurate simulation
• Synthesizable model
• ASIC entry point
• VHLD, Verilog

Figure 6 – Differences between abstraction levels

An increase of 10x in the speed of simulation is expected from CA to TLM and more than 1000x comparing RTL to TLM. The result of this implantation was the acceleration in the development of IPs, with early correction of bugs and simulation of several characteristics of the IP, mainly the functional behavior. Today, great effort is directed to develop ways to perform timed behavior and power analysis in TLM, further improving early architecture exploration.

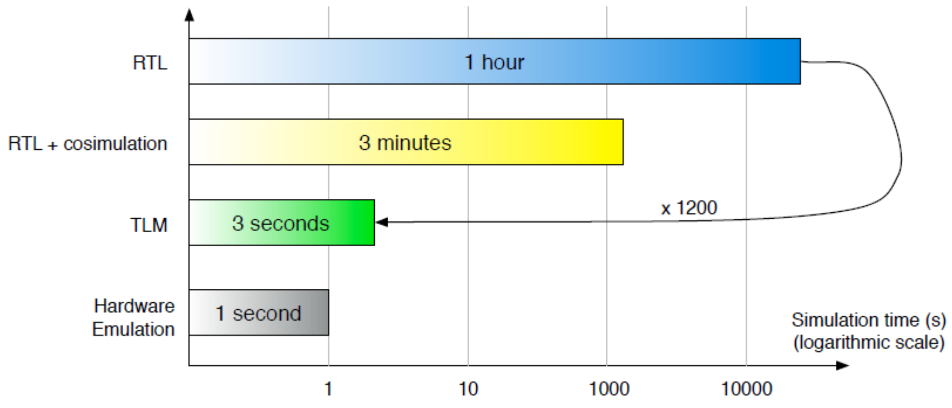


Figure 7 – Simulation time thought different abstraction levels

2.3.1 SystemC

SystemC is an industry standard language, created by a consortium of CAD vendors in 1999 and standardized by IEEE in 2005. The need of efficient simulation, modular design and core reuse lead to use C++ language due to its popularity and efficiency. In fact, SystemC is implemented as a set of C++ classes for digital systems modeling, in other words an open source C++ library. The choice of distribution as an open source library comes from the fact that it should be supported by CAD vendors, IP and software providers.

One of the main differences between common developing languages and HDL languages is the notion of parallelism, a real need in SoC simulation. In counterpart, SystemC uses description parallelism and provides an event-driven simulation core within the library, providing accurate cycle-less simulation.

Nowadays SystemC is widely used and supported in the industry. The implementation in library form, with open standards and C++ language has most benefited the expansion of the language. There are several tools for VHDL and Verilog co-simulation support, with also visualization, waveforms and simulation tools. The benefit from C++ is also remarkable, providing reuse of the language compilers and debuggers.

2.3.2 Registers bank modeling on TLM

The design of registers bank in TLM is well supported by STMicroelectronics development teams, for a specific development kit called *tlm_register_bank* was created with this purpose. Using C++ classes they model the bank, registers and fields, including their attributes. There are several advantages of using this devkit:

- Easy description of the register bank: manually declaring registers or using a register map file which can be automatically created by converter tools (*tlm_skeleton*) using the IP-XACT description.
- Versatile Side-Effects API: provides easy definition of special behaviors between registers and fields with also support to commonly used side-effects. Any combination of attribute between registers and fields is valid respecting the rule of the same size of bits for both.
- Standard messaging: Debug, warning and error messages for easy debug of the register bank.
- Protocol independent implementation: the bank does not depend on the communication protocol which can be changed without problems.

Besides the well-covered implementation of the devkit most of the company developers were already familiarized with, these were the main motivations to use this devkit as base model for the upgrade of *spirit2regtest*.

3. Related Work

It is important to note that the generation of eSW for register test is a very specific part of the whole validation process, that is why there are no available tools in the market performing the exact same function as we expected for the project. Knowing this we decided to focus in the analysis of the existing process of the company that was already being used for more than a year, but still not covering all the users' requirements. We continue this text with the analysis of *spirit2regtest.*

3.1 Spirit2regtest tool analysis

Spirit2regtest is a converter tool which automatically generates the eSW for registers test. It takes the IP-XACT description of the IP as input, parses the component memory map and serializes this information creating tests for each register in a chain. The output is a collection of C files including header files with IP memory map information, support files for standalone test and the test environment itself.

Internally, the algorithm of the tool is globally quite simple, the complexity comes from the amount of allowed parameters for the user and also the amount of information per register. It starts by parsing the description file input using XML::Simple parser, a free software library. XML::Simple works by parsing an XML file and returning the data within it as a Perl hash reference. Within this hash, elements from the original XML file play the role of keys, and the data between them takes the role of values. Once XML::Simple has processed an XML file, the content within the XML file can then be retrieved using standard Perl array notation. Next the tool verifies each element of the component memory map generating code according to user parameters and register attributes.

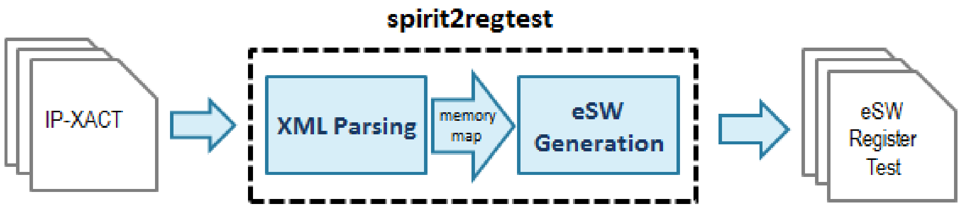


Figure 8 – *spirit2regtest* conversion process

The analyzed version generates a chain of tests for each register according to its access rights. It begins by testing the initial value of the register (if it is readable), testing the access right and finally an additional test in case of advanced attributes on the IP-XACT description. Besides, thanks to a ST vendor extension, registers described with "SideEffect" does not have any test generated, instead the tool places a warning message. The table below (Table 1) summarizes the possible generated tests according to attributes:

Test name	Description	Scope
Initial Value test	Verify registers initial value	Every readable register
Read-Write test	Walking ones test	RW
Read-Write light test	Pattern test	RW
Read-Only test	Checks RO att.	RO
Write-Only test	Checks WO att.	WO
Read-WriteOnce test	Checks RWOnce att.	RWOnce
WriteOnce test	Checks WOnce att.	WOnce
Alias test	Check registers address	RW
ModifiedWriteValue test	Checks modified write value attributes of IP-XACT	BCLR, BSER, ZCLR, ZSET, BTOGGLE, ZTOGGLE, WCLR, WSET
ReadAction test	Checks read action parameter of IP-XACT	RCLR, RSET

Table 1 – Generated tests of analyzed version

Finally the tool also generates the main.c file with an entry point for the test chain, *makefile* and configuration files for standalone execution of tests. This is the default operation of the tool, but it is also possible to generate only header files or specific files for some ST divisions using command line parameters. The generated files are portable from TLM to Silicon without modification. They are summarized below:

- *<ip_name>.h*: contains C macros with information of each register of the bank which will be used by the tests.
- *<ip_name>_test.c/.h*: contain the test chain for the register bank. Each register has its own test function with specific methods for its fields and attributes.
- *s2rt_regsutil.c/.h*: contain generic functions for basic register test which are used by other generated test files.
- *s2rt_config.h*: contains configuration setting for the execution of the generated test, may be modified according to user environment settings.
- *main.c:* Main file which declares the local memory map and calls the test chain.
- *<ip_name>_standalone.h*: This file is used to simulate a test in the host (not embedded in a TLM platform).
- *Makefile.<ip_name>*: Makefile used to build the generated test for self-test mode. Can also be used as a model for compilation of the files.

3.2 Missing features

Thought this analysis we detected two main aspects of the tool that must be discussed. At first, there is no distinction between RTL and TLM verification. Tests as walking ones and alias are not suitable for TLM since their objective is to identify manufacture issues not explored in TLM level, these tests must be skipped in case of TLM verification. Secondly, the tool does not provide any support functions in case of register side-effect, forcing the user to code the test by himself. Hard coded tests increase the chances of bugs and development time.

3.3 Users requirements survey

Before finishing the analysis, it was also performed a survey between users of the tool in order to identify weak points of the generated code and difficulties of the users when using it. The major issue reported by the users was the difficulty performing a consistent test without errors caused by side-effects. The current tool version does not make any distinction between side-effect registers and common registers declaring their tests sequentially in the description order. This organization forces the user to search over the test chain for the registers that are causing side-effects and change the order of execution of the tests, testing the target before the side-effect register. For complex IP's with large registers bank this is a hard and repetitive task. Finally, some users also replied about the known problem of lack of support code for side-effect registers, giving examples of algorithms which have been commonly used on the tests.

3.4 Comparison between processes

The analysis performed over the flow was important to understand the differences and problems that were disturbing users. First, there was no way to completely automate this process since registers can have very specific behaviors, however, the commonly used side-effects can absolutely be standardized, *tlm_register_bank* is a proof of that. Next, IP-XACT still does not have enough information to describe commonly used side-effects, which is why both sides of the verification process do not automatically generate the code for this. Finally, *spirit2regtest* does not have any development support for advanced behaviors. The table below (Table 2) summarizes these issues according to register attributes categories.

	eSW side		TLM-IP side	
	Support	Auto-generate	Support	Auto-generate
Basic Attributes	OK	OK	OK	OK
Adv. Attributes (self-applied)	OK	OK	OK	OK
Adv. Attributes (with target)	**NO**	**NO**	OK	**NO**

Table 2 – Summary of analyzed verification flow

3.5 Upgrade Specifications

Finally, thanks to this analysis, a precise list of specifications for the project could be done. In meeting with *spirit2regtest*'s main developer, Mr. Kamlesh Pathak and this project industrial advisor, Mr. Laurent Bernard, we decided to perform the tool upgrade in two stages giving time to perform half-way validation between them since the second part relies on the correct operation of the first.

The first part of the improvement was to upgrade the tool, improving the generated code according to users demand and adding support to advanced attributes. Precisely the following points:

- Simplify code generated for main.c file and include more support for TLM tests;
- Create simple entry point for registers with side-effect;
- Develop standard functions for common side-effects (advanced attributes);
- Develop generic functions for custom side-effects;

The second part was to improve the automation of the process including new side-effect covering for register test. Two different options were discussed: providing an extra input file with only the effects descriptions or improve the IP-XACT descriptions through vendor parameters. We decided to apply the second one since the descriptions were actually in use and probably the attributes would naturally be included after in the next official schema releases. Finally, with enough information available the tool could be upgraded to parse and create more complete tests:

- Define new IP-XACT parameters for common side-effects;
- Upgrade *spirit2regtest* with automatic generation for common side-effects;

4. Project Development

This chapter describes the implementation of the proposed solutions for the problems described in the last chapters. The implementation was divided in two main stages, first the improvement of the tool itself followed by the automation of the process.

4.1 Tool improvement

The first task before start upgrading the tool was to understand specific constraints of eSW and prepare a proper test platform. For that the first weeks were spent in research over embedded software constraints, tests strategy and also comprehension of tool's legacy code.

A platform for tests was constructed in order to validate the tool while performing the improvements. The design under test (DUT) IP-XACT description was created, a simple target component with a large register bank where had been declared at least a register for each possible attribute, as well as extra registers to use as targets for side-effects. The TLM platform was generated using ST-flow tools and consisted basically on: a native wrapper to simulate the CPU, a memory component to store the eSW, the DUT and a bus to interconnect the components. This process was done several times during the development to re-create the DUT with different register bank configurations. The test eSW generated by *spirit2regtest* was executed in the CPU component in HCE mode.

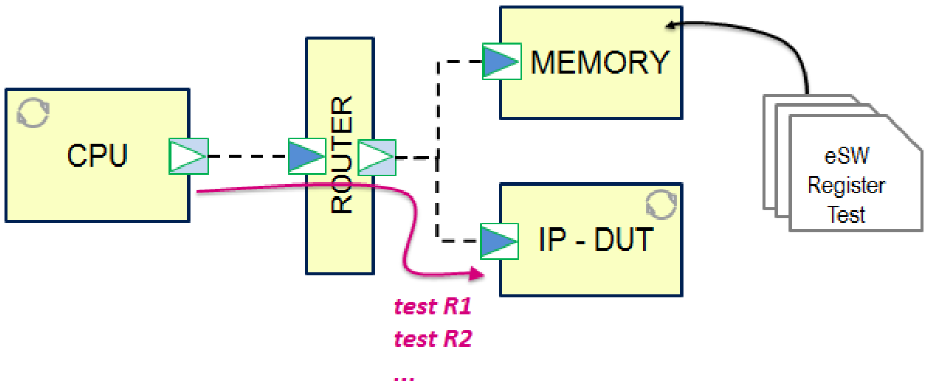


Figure 9 – Simple platform for test during the development

4.1.1 Structural modifications

Before starting developing the support functions, some structural modifications were performed over the tool regarding user requirements. Initially some small changes were done over the eSW configuration files to provide more suitable tests for TLM, avoiding walking ones and alias test. For *main.c* file the objective was to let it more neutral, therefore, output and verbose functions were moved to other files leaving only the main call for the test chain.

To improve the interface between users and the generated eSW it was necessary to modify the structure of the current implementation. In the improved file architecture, special registers were separated from the common registers in a new file called *<ip_name>_side_effect_test.c/h,* providing a new test chain file with only the registers declared with special behaviors. The new chain of tests for special registers is called at the end of the basic test chain therefore common registers were already tested and possible side-effects will not hamper their test anymore.

4.1.2 Side-effects utilities

With friendlier test organization for the user, it was time to provide the tests development support for registers with side-effects. For this purpose two new files were added to the architecture following the same style of *s2rt_regsutil.c/h*, the files were called *s2rt_regsutil_side_effect.c/h*.

These new files contain basically two kinds of support, functions for commonly used side-effects and functions for custom side-effects. For advanced attributes (commonly used side-effects), complete test functions were provided allowing the users to set up tests for registers and fields only pointing them through function parameters. For custom side-effects, generic functions were also provided, with utilities that can help users to develop their own tests as read and write directly from field, for example.

The main difficulty of this task was to provide functions respecting embedded code constraints, such as fixed data types and right bitwise operations. The use of masks representing register bit fields was chosen as the best option since it is possible to perform any verification with only this information. The reads and writes performed in embedded software world must always respect registers data type to ensure correct operation that is why most of the developed functions have the register size as a parameter.

User compatible interface was also taken into account since the success of use relies directly on the development of friendly user APIs. Unfortunately C language does not provide the elegance of oriented object languages which allow much more flexible constructions of functions prototypes. Even so, the support functions were designed to always follow the same pattern, using basically always the same quantity of information, as the following example:

```
uint32_t BCLR_register_test(  src_addr, src_size, sfld_width, sfld_offset
                              tgt_addr, tgt_size, tfld_width, tfld_offset);
```

This function performs the bit clear behavior (*BCLR*) test between the source (*src, sfld*) and target (*tgt, tfld*) fields. The choice for field and width instead of directly the mask was made to avoid user errors as composed (e.g. 11100111 declares 2 fields of 3 bits) and bad calculated masks. The return of all test functions is the number of errors occurred during the test. The whole set of parameters can be encountered in the *<ip>.h* file in the form of C *#define* macros, the user does not need to remember a bunch of numbers but only the source and target names. A real example of use can be seen in the snippet below:

```
Err_count += BCLR_register_test( base_addr + REG1_OFFSET, REG1_SIZE,
                                 REG1_F1_WIDTH, REG1_F1_OFFSET,
                                 base_addr + REG2_OFFSET, REG2_SIZE,
                                 REG2_F1_WIDTH, REG2_F1_OFFSET);
```

In this example, this function call will perform the verification of the bit clear operation between the field “F1" of the register “REG1” and the field “F1” of the register “REG2”. At this point of the project every function call was coded manually, however the whole set of parameter in this function is available within the IP description and the automation can be done, as we will see in the next chapter (4.2).

For the functions test strategy, it was chosen a two-phase test that verifies the correct operation of the source. Each function performs a **not effective** test, writing the value that **must not** cause the effect in the target register and verifying the target value that must contain the same value. Next each function performs an **effective** test, writing the value that **must** cause the effect in the target register and verifying that the target value has changed. For instance, the BCLR test algorithm is described below:

<u>Bit Clear Test</u>:

1. Save source and target original values;
2. Replace (write) target value to 0xFFFF…;
3. Write 0x0000… in source (not effective test);
4. Verify that target value still the same;
5. Write 0xFFFF… in source (effective value);
6. Verify that target value changed to 0x0000…;
7. Restore source and field original values;

On the last released version there were available 22 test functions and 6 general-purpose functions for the user. For further information, the complete set of functions is available on the user manual and can be seen in the appendix (App D).

4.1.3 First Release

Comparing with the previous tool release, here is the list of the main changes in the tool:

- Neutral *main.c* file;
- More support for TLM verification;
- Separation between normal and SE registers;
- Support functions for advanced attributes;
- Support functions for custom side-effects;

An important remark is that all the new features of the tool are controlled by a command line argument (-*side_effect_management*) in order to maintain compatibility for old version users. The new architecture of the generated files is illustrated the image below (Figure 10).

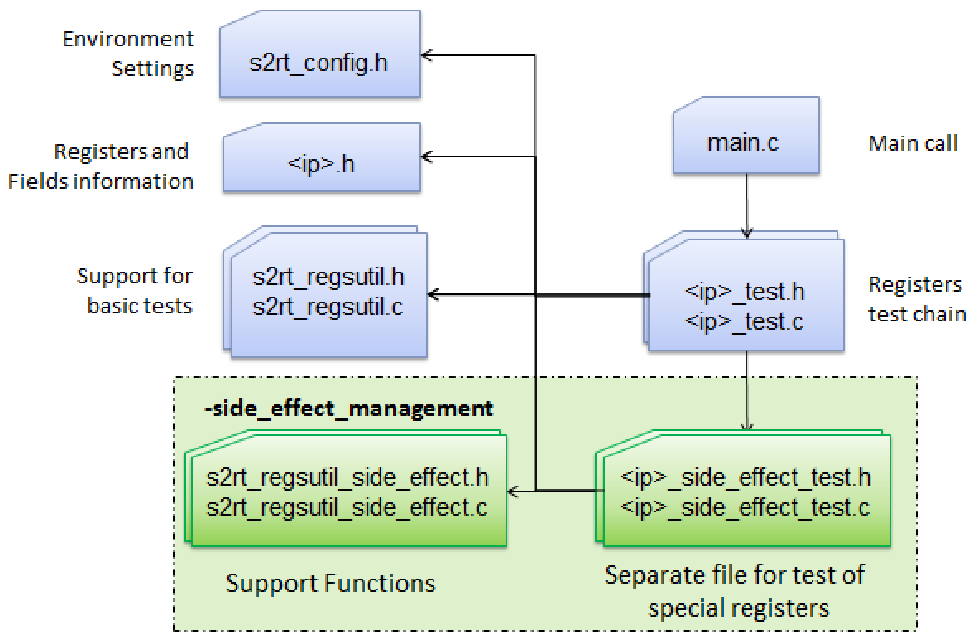


Figure 10 – New file architecture

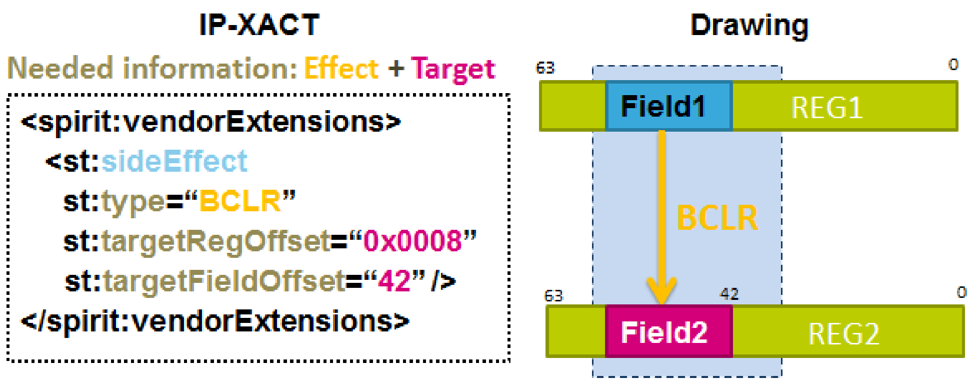
4.2 *Process automation*

This chapter describes the second stage of the project implementation where we increase the automated code generation capability of *spirit2regtest*. We start by the improvement of IP-XACT descriptions followed by the upgrade of the tool to parse this new information, finally providing complete support for advanced attributes.

4.2.1 New IP-XACT parameters design

The first task required for side-effect test automation was the design of new IP-XACT extensions in order to provide the missing information for common side-effects. As presented in the IP-XACT chapter the missing information were the <u>effect</u> and <u>target field</u>, without these key information it is not possible to automatically generate the tests.

With the help of team's IP-XACT specialist we have designed vendor extensions to improve the standard schema. The extensions must be described inside the field structure and can be used multiple times as needed. In other words, a register can have multiple fields and a field can have multiple side-effects, for example, a field with maximum and minimum value depending of two other fields. Each description will have three attributes: <u>effect</u>, <u>target register offset</u> and <u>target field offset</u>. We decided to put only these three parameters to avoid overloading the descriptions, the target complementary information can be encountered in the IP description. Finally, we decided to use values instead of names because the names change more frequently than their offsets in IP design. The diagram below (Figure 11) shows the designed vendor extensions for side-effect description.



```
<spirit:vendorExtensions>
  <st:sideEffect
    st:type="BCLR"
    st:targetRegOffset="0x0008"
    st:targetFieldOffset="42" />
</spirit:vendorExtensions>
```

Figure 11 – Vendor extensions for advanced attributes

4.2.2 Process Automation

Once the improved IP-XACT description was defined, it was time to upgrade *spirit2regtest* code generation. Since the basic registers test were already being automatically created, the declaration of the advanced tests is limited to registers described as having special behavior. The tool parses the vendor extensions and automatically declares the calls to the support functions created in the first implementation stage of the project. The needed parameters for the field are fetched in the XML description using the register and field offset. The tool begins by searching for the register node which has his unique address offset, there it takes the register size. Next it searches for the field node which also has a unique bit field offset, there it takes the field width. With these data the tool must only declare the right test function call, according to the effect value in the vendor parameter. In case of unknown effect or any problem searching for the field (e.g. bad address offset) the tool will not generate any test, but a warning instead.

4.2.3 Second Release

With this improved version the user does not need to care with most of registers side-effects. Using the designed parameters the user needs only to declare the known effects in the IP-XACT description, the tool will automatically create the tests. For unknown effects the tool will create the test entry point and put a message indicating side-effect existence, the user only need to open the file with the special registers and code the test with help of the support functions. For the second release of this project we can remark the following new features:

- Supports new vendor extensions;
- Automatically generate test for registers with advanced attributes;

5. Results and Discussion

Here are presented the obtained results in the last stage of the project where was performed the evaluation of the last release of the software. Furthermore, a personal discussion is also presented were we talk about the work done and the final product.

5.1 Basic Tests and Non-regression Tests

Most of the validation of this work was done during the development, performing black-box tests with each test chain implemented. Besides, all the code respects C99 rules and good practice rules imposed by the company. To ensure compatibility and non-regression of the new versions for the users, there were also performed TSP (Test-Suite Processing) tests with a pre-defined suite of tests, which generates code for several IP descriptions and compares with golden models. Every release of the software passed through all the TSP tests without error.

5.2 Test on Interrupt Controller with Power PC

A test with a real complex IP was performed in order to validate the final release and ensure that the software was still portable through different platforms. The major difference of this test in relation with the development test was that this platform used a different kind of processor, precisely a Power Architecture (also known as PowerPC) processor from Freescale®, simulated through an ISS. The compiler was provided by WindRiver® company.

The design under test was an Interrupt Controller with about 1070 registers in his bank. The IP has this elevated number of registers because each interruption have a priority, the controller support up to 1024 interruptions so having 1024 priority registers. Finally, it was verified also that several registers have special behaviors as side-effects, or being flags to interrupts, a suitable IP to test the tool.

For this test the TLM model of the controller was already configured. The platform was generated and the ISS was included using an UART communication channel to provide the processor output traces. Using a tool called TLM Device bound to the processor UART it was possible to configure the data output and visualize the verbal output of the generated software.

After analysis of the IP-XACT description it was verified that it did not describe the special behaviors in the registers, so this information was added into the description using the designed vendor extensions. Finally, using the latest *spirit2regtest* version, the code was generated and compiled for the PowerPC processor, being executed after and performing the test correctly.

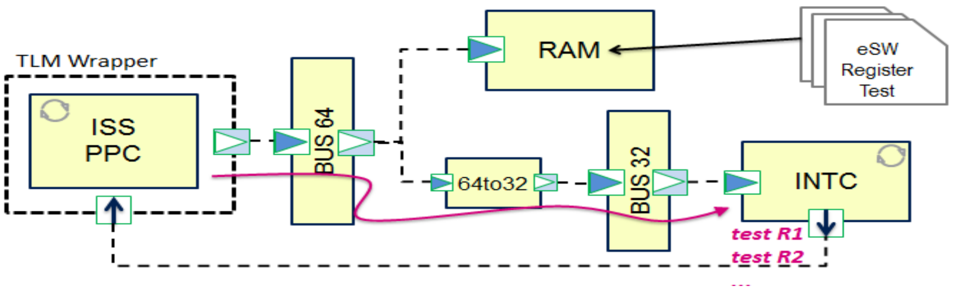


Figure 12 – Industrial test platform

5.3 Acceptance and Team Feedback

After an introductory presentation and demonstration, the team liked the work performed over the tool, also the new designed vendor extensions will be included on other company softwares. Since we had at least two releases of the tool, after three months of use there were no request concerning the tool, this can be seen as a good thing since there were no reported errors neither misunderstanding problems.

5.4 Solution Limitations

Unfortunately the choice of C as language for the generated code drives to a less elegant solution. With object-oriented languages, as C++ for example, there are many more features to be explored as method overload and inheritance which could provide a better interface for the user. Even so there are few processor compilers that support high level languages. C language was chosen since it is largely more compatible.

Another limitation is that the tool generates code for several possible simulations, with the possibility to enable or disable specific tests. In other words, depending of the desired tests, great amount of dead code remains inside the software. Depending on the compiler optimizations this can hamper the process raising the size of the binary files, the needed amount of memory and also the test speed. Hopefully most of the compilers today perform this kind of optimizations, suppressing dead and unreachable code, the test will be performed at its maximum efficiency.

5.5 Perspectives

We hope that this project will be used and improved along the years since more commonly used side-effects can appear. With the developed work in place, implementing more support functions is not a hard task. Besides, IP-XACT standard schemas are also evolving year by year, this work can be a contribution to the standardization of commonly used side-effects, officially including the vendor extensions in some part of the next schema release.

For STMicroelectronics verification flow, this work can be used to improve not only test generation, but also the TLM virtual platform generation since now there is sufficient information on the IP-XACT schema.

6. Conclusion

The industry is on track to meet the requirements of automation. Through this project it was possible to get in touch with today's semiconductors industry processes and real needs. It was also very interesting to see real semiconductors design flow, integrating several systems, databases and methods in order to raise production speed.

With progressive work, all the project objectives were reached and the final product is already on use. After meticulous analysis of the existent tool, the possibilities and needs of the test process, we performed the development over the flow upgrading both description standard and test generation tool, finally validating it with tests for real IPs. The improved register bank test process benefits the company helping users on TLM and RTL, validation and verification of IPs, saving time to other important activities. It now generates much more complete software for register testing, highlighting specific behaviors of registers and offering a complete support for developing their tests.

This project was composed of mixed activities on SoC design in different abstraction levels, working with hardware and software development. There were specification level tasks such as the improvement of IP-XACT descriptions, TLM level tasks as the creation and generation of virtual platforms for register testing, finally, automation tasks such as the improvement of register test generation tool. It also helped me develop my knowledge of a small, but essential part of chip design. Work with TLM can be a grateful job for those who love the logic of digital systems but did not like to enter deeply on physic low representation levels. TLM allows the production of virtual SoC with the elegance and simplicity of high level languages. Another gain was the experience with embedded software, especially with their constraints that, at my point of view, requires much more caution developing since they may be executed in different processors, and normally without any operational system support.

From a personal point of view, this opportunity was especially important for me considering the scope of international studies. Besides the scientific and technic improvement of the knowledge we cannot forget the improvement of basic human skills as sociability, communication and organization. The work was performed within an international company environment, featuring communication with people from different cultures around the world. In the same way because the wide technology field knowledge which requires good synthesis skills to understand, catch the essential information, simplify, exemplify and share. I'm very happy to say that this was my best work experience so far and that I feel comfortable moving forward working in this knowledge area.

Bibliography

- STMicroelectronics, "Who We Are", Mar. 2013. Web. 27 Feb. 2013. <http://www.st.com/st-web-ui/active/en/about_st/st_company_overview.html>

- iSuppli, “Qualcomm Rides Wireless Wave to Take Third Place in Global Semiconductor Market in 2012", Dec. 2012. Web. 27 Feb. 2013. <http://www.isuppli.com/Semiconductor-Value-Chain/News/Pages/Qualcomm-Rides-Wireless-Wave-to-Take-Third-Place-in-Global-Semiconductor-Market-in-2012.aspx>

- iSuppli, “Worldwide 2005 top 20 semiconductor market share ranking", Mar. 2006. Web. 27 Feb. 2013. <http://i.cmpnet.com/siliconstrategies/2006/03/isupplitables.gif>

- Accellera , “IP-XACT Technical Committee“, Unkwnown. Web. 16 May 2013. <http://www.accellera.org/activities/committees/ip-xact/>

- Gartner , “Worldwide Semiconductor Revenue Declined 2.6 Percent in 2012”, STAMFORD, Conn., Apr. 3, 2013. <http://www.gartner.com/newsroom/id/2405215 >

Appendix

A. Simple Register Description in IP-XACT

```
<spirit:register>
  <spirit:name>MYIP_SER</spirit:name>
  <spirit:description>A read-write 8 bits register
  </spirit:description>
  <spirit:addressOffset>0x003B</spirit:addressOffset>
  <spirit:size>8</spirit:size>
  <spirit:access>read-write</spirit:access>
  <spirit:reset>
    <spirit:value>0xFF</spirit:value>
  </spirit:reset>
  <spirit:field>
    <spirit:name>ZSET</spirit:name>
    <spirit:description>Zero set advanced attribute test
    </spirit:description>
    <spirit:bitOffset>4</spirit:bitOffset>
    <spirit:bitWidth>2</spirit:bitWidth>
    <spirit:access>read-write</spirit:access>
  </spirit:field>
</spirit:register>
```

B. Improved Register Description

```
<spirit:register>
  <spirit:name>MYIP_SER</spirit:name>
  <spirit:description>A read-write 8 bits register
  </spirit:description>
  <spirit:addressOffset>0x003B</spirit:addressOffset>
  <spirit:size>8</spirit:size>
  <spirit:access>read-write</spirit:access>
  <spirit:reset>
    <spirit:value>0xFF</spirit:value>
  </spirit:reset>
  <spirit:field>
    <spirit:name>ZSET</spirit:name>
    <spirit:description>Zero set advanced attribute test
    </spirit:description>
    <spirit:bitOffset>4</spirit:bitOffset>
    <spirit:bitWidth>2</spirit:bitWidth>
    <spirit:access>read-write</spirit:access>
    <spirit:vendorExtensions>
      <st:sideEffect
      st:type="ZSET"
      st:targetRegOffset="0x0054"
      st:targetFieldOffset="4"/>
    </spirit:vendorExtensions>
  </spirit:field>
</spirit:register>
```

C. Generated Test

```
/*!
* \brief
* myip _ser_register_test : MYIP_SER register test function
* Scope : testMemMap/testRegisters/MYIP_SER
* Purpose : Test S2RT_SER register and return the test status
*/
uint32_t   myip_ser_register_test(ADDRESS_TYPE   address,   TEST_TYPE
test) {
   /** variable declaration */
   uint32_t errorNbr;
   uint8_t pattern;
   uint8_t index;
   uint8_t expected_value;
   uint8_t current_value;
   /** variable initialisation */
   …

   /*!
   * \brief
   * Initial Value test for MYIP_SER register
   */
   if (test & TEST_RESET) {
        errorNbr += VALregister_test_8(address,
                    myip_MYIP_SER_RESET_VALUE,
                    ((myip_MYIP_SER_RWMASK | myip_MYIP_SER_ROMASK)
                    & (~myip_MYIP_SER_RAMASK)));
   }

   /*!
   * \brief
   * ZSET test for S2RT_SER register
   */
   if (test & TEST_SPREG) {
     errorNbr += ZSET_register_test(myip_BASE_ADDRESS +
                myip_MYIP_SER_OFFSET,
   myip_MYIP_SER_SIZE,
   myip_MYIP_SER_ZSET_WIDTH,
   myip_MYIP_SER_ZSET_OFFSET,
   myip_BASE_ADDRESS +
   myip_MYIP_TG16_OFFSET,
   myip_MYIP_TG16_SIZE,
   myip_MYIP_TG16_F2_WIDTH,
   myip_MYIP_TG16_F2_OFFSET);
   }
   return errorNbr;
}
```

D. User Manual Section

SIDE EFFECT SUPPORT FOR REGISTER TEST

Spirit2regtest support for side-effects is activated using **-side-effect-management** option. The tool provides additional files with advanced test functions for the behaviors and also automatically generates tests for registers which contains the vendor extension defined below.

IP-XACT ST Vendor Extension for side-effects

This vendor extension was designed to provide IP-XACT registers fields' side-effects declaration. It must be declared within the field node of the source register (the register that produces the side-effect) and has three parameters:

- **type**: register Effect, can be one of the following: BSET, BCLR, ZSET, ZCLR, BTOGGLE, ZTOGGLE, WSET, WCLR, RSET, RCLR, MAX, MIN, STEP, SBO, SBZ.
- **targetRegOffset:** hexadecimal value for the register target offset.
- **targetFieldOffset:** decimal value for the target field offset within the target register.

If the declared behavior (type) is not recognized by the tool the test will not be generated, only the test function entry. The snippet below shows the parameter in IP-XACT format:

```
<spirit:vendorExtensions>
     <st:sideEffect st:type="EFFECT"
      st:targetRegOffset="0xXXXX"
      st:targetFieldOffset="XX"/>
</spirit:vendorExtensions>
```

NOTE: Do not forget to declare the st namespace in the document header:

*<spirit:component **xmlns:st="http://www.st.com/XMLSchema/SPIRIT"**>*

Multiple side-effects can be declared over a field, e.g. the user can add Maximum and Minimum value constraint on a single field (where the constraint value is on another 2 fields).

Advanced Test Functions Description (s2rt_regutils_side_effect)

S2rt_regsutil_side_effect file contains a set of advanced test functions and utilities for test development. The return for the test functions is the number of errors occurred during the tests. As input the user must provide basic information about the register/field and also for the target when it exists. These functions are divided in 4 groups:

Bitwise operations between two registers fields

These functions perform the side-effect test between two fields, verifying that the source field changes the value of the target field correctly. Each function store the source and target original values at the beginning and restore the values ate the end. The user must provide 4 parameters for each field: *register address, register size*, *source width*, *source offset*.

Each function performs two tests: verifies that the source field really changes the target on true condition, verifies that the source field does not change the target value on false condition.

Test Name	Description
BCLR_register_test	Write 0x00… pattern into source field Check no changes in target bits Write 0xFF… pattern into source field Check clear in target bits
BSET_register_test	Write 0x00… pattern into source field Check no changes in target bits Write 0xFF… pattern into source field Check set in target bits
ZCLR_register_test	Write 0xFF… pattern into source field Check no changes in target bits Write 0x00… pattern into source field Check clear in target bits
ZSET_register_test	Write 0xFF… pattern into source field Check no changes in target bits Write 0x00… pattern into source field Check set in target bits
ZTOGGLE_register_test	Write 0xFF… pattern into source field Check no changes in target bits Write 0x00… pattern into source field Check toggle in target bits
BTOGGLE_register_test	Write 0x00… pattern into source field Check no changes in target bits Write 0xFF… pattern into source field Check toggle in target bits
WCLR_register_test	Read source field Check no changes in target bits Write in source field Check clear in target bits
WSET_register_test	Read source field Check no changes in target bits Write in source field Check clear in target bits

RCLR_register_test	Read source field Check clear in target bits Write in source field Check no changes in target bits
RSET_register_test	Read source field Check set in target bits Write in source field Check no changes in target bits

Table 3: Bitwise operations test functions description

Interdependencies between two registers fields

These functions perform the side-effect test between two fields, verifying the correct operation of the bounded field in relation with his limit/step register. These functions calculate an out of bound value and apply to the field verifying that the value automatically changes.

Test Name	Description
MAX_constant_register_test	Write value greater than constant in field Check that the current field value is the maximum possible
MAX_variable_register_test	Read targeted maximum field Write value greater than maximum Check that the current field value is the maximum possible
MIN_constant_register_test	Write value lower than constant in field Check that the current field value is the minimum possible
MIN_variable_register_test	Read targeted minimum field Write value greater than minimum Check that the current field value is the minimum possible
ConstantBounds_register_test	Perform MAX_constant_test Perform MIN_constant_test
VariableBounds_register_test	Perform MAX_variable_test Perform MIN_variable_test
STEP_constant_register_test	Write out-stepped value (if exists) Check that the current field value is steeped
STEP_variable_register_test	Read targeted step field Write out-stepped value (if exists) Check that the current field value is steeped

Table 4: Interdependence operations test functions description

Restricted Value Attributes

These functions perform the field test which has restrictions to write values. They test each allowed value and also test a not allowed value to confirm the constraint.

Test Name	Description
WRITE_AS_READ_register_test	Write reset value on field Check that field value still the same Write field inverse value Check that field value still the same
ENUM_register_test	Write each enumerated value and check Write a not enumerated value Check that the field value does not change
SBZ_register_test	Check that field accepts only zeros
SBO_register_test	Check that field accepts only ones

Table 5: Restricted value test functions description

General purpose functions

These functions were created to be used as a support for custom cases where the user must code the register test.

Test Name	Description
register_write	Write value on register
register_read	Read register value
field_write	Write value on field (access entire register)
field_read	Read field value (access entire register)
register_expected_behavior_test	Write value in source register Read target register value Compare with expected value
field_expected_behavior_test	Write value in source field Read target field value Compare with expected value

Table 6: General purpose test functions description