SoC Security and Debugging through CR Management and Access

Major Project Report

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology

in

Electronics & Communication Engineering

(VLSI Design)

By

Jani Khyati Ashok

(16MECV09)

Electronics & Communication Engineering Department

Institute of Technology

Nirma University

Ahmedabad-382481

May 2018

SoC Security and Debugging through CR Management and access

Major Project Report

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology
in
Electronics & Communication Engineering

(VLSI Design)

By

Jani Khyati Ashok

(16MECV09)

Under the guidance of

External Project Guide:

Mr. Amar Jituri

Engineering Manager,

Intel Technology India Ltd.,

Bangalore

Internal Project Guide:

Dr. Usha Mehta

Professor in EC Engineering,

Institute of Technology,

Nirma University, Ahmedabad

Electronics & Communication Engineering Department

Institute of Technology

Nirma University

Ahmedabad-382481

May 2018

Declaration

This is to certify that

1. The thesis comprises my original work towards the degree of Master of Technology in VLSI design at Nirma University and has not been submitted elsewhere for a degree.

2. Due acknowledgment has been made in the text to all other material used.

- Jani Khyati Ashok

16MECV09

Certificate

This is to certify that the Major Project entitled **SoC Securiy and Debugging through CR Management and Access** submitted by **Jani Khyati Ashok (16MECV09)**, towards the fulfillment of the requirements for the degree of Masters of Technology in VLSI design, Nirma University, Ahmedabad is the record of work carried out by her under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this Project, to the best of our knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Date: Place: Ahmedabad

Dr. Usha Mehta
Internal Guide,
Professor in EC Engineering,
Institute of Technology,
Nirma University, Ahmedabad

Dr. N.M. Devashrayee
Program coordinator (VLSI),
Professor in EC Engineering,
Institute of Technology,
Nirma University, Ahmedabad

Dr. D. K. Kothari
Head,
EC Engineering Department,
Institute of Technology,
Nirma University, Ahmedabad

Dr. Alka Mahajan
Director,
Institute of Technology,
Nirma University, Ahmedabad

Certificate

This is to certify that the Major Project entitled **Soc Security and Debugging through CR Management Access** submitted by **Jani Khyati Ashok (16MECV09)**, towards the fulfillment of the requirements for the degree of Masters of Technology in VLSI design, Nirma University, Ahmedabad is the record of work carried out by her under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this Project, to the best of our knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Date: Place: Bangalore

Mr. Amar Jituri
Engineering Manager,
Intel Technology India Ltd,
Bangalore.

Acknowledgements

I owe my deepest gratitude to my manager, **Mr. Amar Jituri** whose expertise, understanding, generous guidance and support made it possible for me to work in Intel with great enthusiasm. Words are not enough to thank his kindness.

I am grateful to my mentors, **Mr. Amol Jagtap** and **Mr. Ajith Kalangara**, at Intel Technology for their valuable guidance and support during the project work.

I am highly indebted and thoroughly grateful to **Dr. Usha Mehta** for being an excellent guide and to **Dr. N. M. Devashrayee** and all the faculty members of Nirma for their vision, support and encouragement.

I dedicate my Mtech thesis to my parents for their countless sacrifices and unbounded love which has always been a motivation to strive for excellence.

Last but not the least my friends also deserve a big thank you for being there to bounce off ideas or to just vent out frustration over a cup of tea.

- Jani Khyati Ashok

16MECV09

Abstract

Since the dawn of complex systems, numerous experiments have been carried out to design tools for defining Software/Hardware interfaces of such complex systems and to generate useful artifacts which are investigative procedures that will be used in different downstream levels such as from RDL to RTL , from RTL to Validation(Pre-silicon)and then from validation to testing(Post-silicon). In 2005, a tool called Blueprint was released by Denali which uses a pre-established set of data for describing registers using a format called Register Description Language (RDL).Automatic generation and synchronization of register views for specification, hardware design, software development, verification, and documentation can easily be done by developers using SystemRDL. The development of a standard format for all the data which includes all the debugging features, description of registers and hardware related constraints has been mentioned in SPIRIT. The consortium has proposed an IEEE standard called IP-XACT which is an XML Schema Definition used to store the register and memory map data as needed in our tool.

This project mainly focuses on IP integration process which is more prone to errors at SoC level. Quality checks are performed to debug the RDL related errors. The integration flow is automated through Perl scripts, customized and validated to provide efficient implementation ensuring IP protection as far as SoC security is concerned.

Contents

List of Figures

Chapter 1

Introduction

The companies which builds large complex systems like SoC, develop internal register definition technologies and languages as their system may have tens of thousands of HW/SW registers. The Software interacts with the hardware to provide many functions such as:

- Hardware reset to a known state
- Hardware configuration for performing a specific function
- Getting the hardware status information
- Reading/writing data to/from the hardware

Fundamentally, software can either transfer data or can access the hardware (a write access) or retrieve information from the hardware (a read access). Typically this is done on a processor by transforming software instructions to memory mapped transactions to a target specific hardware with either read or write transactions. The processor can access the generic storage mechanisms, status configuration parameters, or status information via a memory-mapped transaction that can be described generally as Software Accessible Hardware Elements (SAHEs). The implementation can be done with constructs such as memories, registers, or bit fields in hardware.

Control registers (CR) are used in all IP products and semiconductor chips and are thousands in number. System architects, software engineers and hardware developers use this registers which is used to store key parameters that define the chip operation, to develop specific end products. System developers and IP providers have benefits from the SystemRDL Alliance commitment to use

computer-sensible common format for describing registers - which ultimately speeds architecture, design, verification, and documentation for semiconductor chip designs which ultimately reduces Time to Market.

The actual hardware RTL description code ultimately needs the address values of all registers and their access modes. This information is also required by the embedded software components while defining a mapping from a register name to the corresponding memory address.

1.1 Motivation

In the present day SOCs, we are integrating millions of gates in-order to embed more and more functionalities and to get increased performance. There has been an integration on multiple functions on a single chip. But this increased integration can cause a lot of bugs to be found in the design. There has been an increase in reliance on semiconductor Intellectual Property (IP) content, both from other design groups within the same company and third party IP suppliers. This has increased the need and complexity of the quality assurance process. If the bugs are found later in the design cycle such as in the physical design process, it will cause a huge loss to the semiconductor company manufacturing the chip. There have been scenarios of such failures. Hence the RTL signoff process plays a very significant role as we start moving towards the lower technology nodes such as 7nm. The handoff process provides enough number of stages to detect and fix significant design issues in a timely manner with the knowledge of the design engineers, rather than looking into the bugs after moving to the backend stages.

Every year the system-on-chip (SoC) designs released by semiconductor companies are becoming more complex. The number of necessary features and supported protocols is increasing, which causes designs to contain a huge amount of different IPs. This makes the integration flow complex at SoC level and the errors related to RDL collaterals increases. Therefore, debugging of errors and validation of RDL integration at SoC level is very much important to ensure proper implementation and IP protection.

1.2 Objective

To validate and customize the IP RDL integration flow and make it error free at SoC level for efficient implementation and optimization of tool flow as far as SoC security is concerned.The work also involve an automation which extracts the required information from the reports generated from Intel-in house tool to perform checks related to RDL Quality and also testing the specifications to ensure quality releases to consumers.

1.3 Organization of the report

The overall organization of the report is as follows:
Chapter 1 discusses the importance and objectives of the project.
Chapter 2 discusses the important concepts and terminologies which are made use in the project.
Chapter 3 discusses the methodology used to carry out the work.
Chapter 4 discusses the errors debugged and the extra checks performed.
Chapter 5 discusses the algorithms used to test the specifications
Chapter 6 is all about Conclusion and Future scope.

Chapter 2

Literature Survey

Cost of SoC design and risk involved in the design are increasing because competitive pressures at global level are causing complexity of system to grow exponentially. Moore's law also places pressure on system design from the other direction by making more and more transistors available to the designers every year. A SoC consists of the most sophisticated and important features with accurate functional specifications and a plethora of requirements. At first step, the complexity of the design is huge. In addition to it, verification of the requirement and specifications is also a huge challenge. One needs to have the knowledge of the complete design flow, its verification and clean-up activities. Before understanding the need of the handoff process, it is essential to understand the flow of the SoC design and where the handoff process comes into picture in the flow. There are certain important terminologies which one should know to understand the process. The typical flow of a SoC RTL signoff process is as shown in the Fig:2.1

The entire set of activities performed by the front end team and the physical design team are part of a SoC design flow. The SoC design flow which is being followed in a semiconductor industry is a very mature and solid process. The complete flow and the various steps which have been followed in the flow have proven to be both robust and practical in todays multimillion gate chips. Each and every step of the SoC design flow has a dedicated EDA tool involved in it which is robust in covering the activities for that specific task.

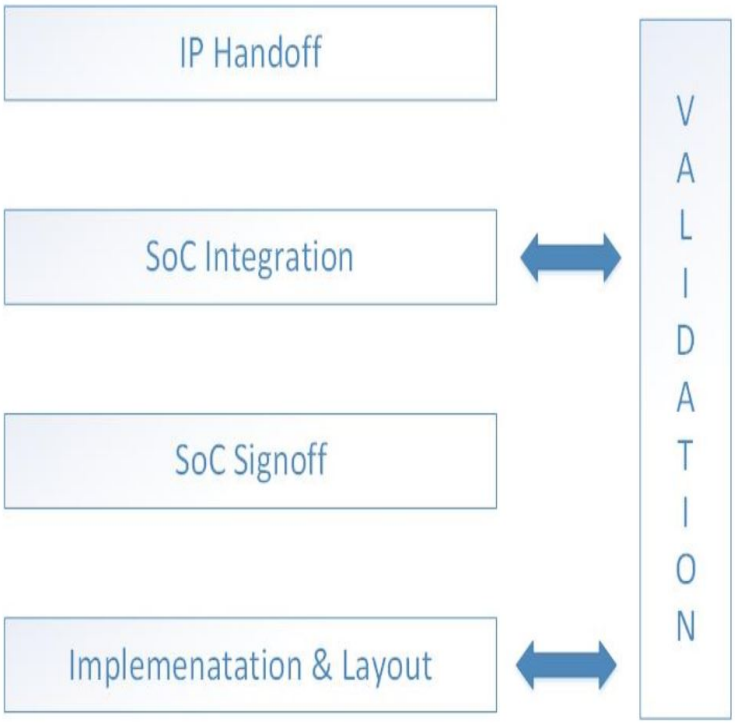


Figure 2.1: SoC RTL Handoff

2.1 SoC Design Methodology

There are many names associated with SoC plan techniques, every one of them allude pretty much to a similar objective: outlining equipment and programming in a similar structure to such an extent that the originator can rapidly deliver the most effective execution for a given framework specification. The term much of the time utilized today is framework level plan, the bland plan of framework level outline is the accompanying:

1. Infer equipment segments and programming segments from the specification,
2. Delineate programming part on the equipment segments,
3. Model the subsequent usage, and
4. Potentially give changes at some phase of the plan if the performance or cost is not agreeable.

There is a worldwide concurrence on the way that abnormal state specifications are very valuable to decrease the plan time. Many plan structures have been proposed for framework level SoC

outline by Gajski. The possibility of refinement of the first specification down to equipment is available in numerous system was once observed as the arrangement yet gives an impression of being a very troublesome issue .Amid the 1990s, the entry of Intellectual Properties (IP) presented a reasonable refinement between circuit fabrication and circuit configuration, driving a few organizations to focus on IP configuration (ARM). Then again, IP-based plan and later platform-based outline propose to begin from settled (or parameterizable) equipment library to diminish the outline space investigation stage. Change in reenactment procedures licenses, with the utilization of systemC dialect, to have an approach between the two by utilizing virtual models.

2.1.1 Platform-based design methodology

SoC designers are considering more flexible implementation methods, which can be modified quickly and effectively. Moreover SoC market demands SoCs which can be used for multiple applications, so the number of IP cores per SoC are increasing. All these factors lead to platform based design methodology which effectively implements reusability and configurability. Platform is actually an abstraction which has all lower level refinements. That is to develop a family of SoC which are similar to each other only differing in few components but are constructed on same platform. A desired product SoC then can be obtained by deriving architecture from the platform instance by replacing few components or reconfiguring the parameters according to need.

2.1.2 SoC Design Reuse methodology

Reusable methodology essentially uses predesigned and preverified IP core. Reuse is considered foundation of SoC design, as it allows the design of complex SoC to meet the rigorous time to market, quality and productivity. To employ reusable methodology is our new challenge. As IP reuse methodology is becoming more advanced, numerous commercial IP reuse tools employing platform-based SoC design methodology are developed. These tools supply designers a standard environment to perform platform-based design with IP reuse methodology. They improve the efficiency and reliability of IP creation and platform integration.

2.1.3 Reusable IP

According to Jerinic Muller (2004) in today's fast developing technology, IP reusability plays important role for fast and efficient integration of SoC. The level of complexity of SoC now a days leads to many challenges while chip designing. To develop a high quality SoC with minimal cost and efforts and deliver it within specified time to market is very crucial. Most of the times a product is developed from its predecessors with more features and better performance. To decrease re-development time, effort and cost IP-based reuse methodology is used. IP (Intellectual properties) are usually parameterized in order to facilitate their use in different applications by configuring them according to user specifications. IP re-usability is an attractive alternative for conventional method of "designing from scratch" since it requires less design efforts and it has been used extensively in soc design. It reduces design and integration efforts but designing and verifying a parameterized IP still remains a challenge today. As IP becomes more generic, new problems arise in verification and integration. An ideal reusable IP should be able to give constant performance across multiple applications and should provide proper interfaces. IP blocks with configurable timing, power and area enable SoC integrator to apply the trade-off that provides for the needs of the application. By adopting an IP reuse methodology, we can implement system modules directly without designing all the modules conventionally. IP reuse method effiectually improves the design efficiency and reduces market risk by using already tested IPs, reduces research costs and reduces development cycle. It allows re-use of previous designs and also reuse of third party IPs.

2.1.4 Design Resuse

The reusable components or IP blocks are nothing but synthesizable RTL (soft cores) or layout level design (also called hardcore). In system-on-chip (SoC) design pre-designed and pretested blocks, IPs are put together on single chip. SoC design can achieve large productivity in shorter time. The aim of soc design is to integrate IP blocks on single chip to attain complex functionality. Design reuse does not come easily. Design reuse should be kept in mind while developing components. It is necessary to find and evaluate reusable components fitting product requirements, and the selected components then integrated to provide for the desired SoC functionality. Integration

may require system to adapt to the functionality of components and corresponding interfaces. All these tasks need robust support from exact tools and methodologies, so that reuse is successful. Standards must be defined for encouraging reuse in all lengths of the design process.

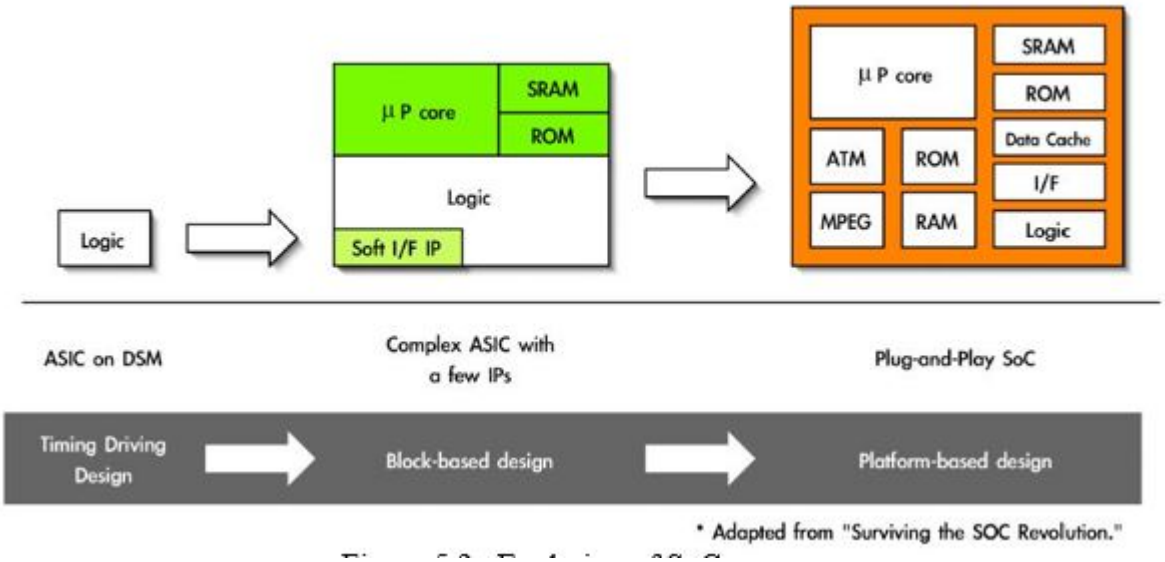


Figure 2.2: Evolution of SoC

The integration of SoC involves connecting such IP blocks, adding design for testability features, verifying and validating the complete soc as a single unit. Derivative designs can be generated easily from architectural designs. The motivation for this methodology is smaller form factor, reduction in power and reduction in overall cost. Overall productivity gain of SoC is measured with reusability of components and integration complexity. Industry wide standards for buses, interfaces, IP exchange formats, documentation, IP protection, and test wrappers have been developed and standardized. Main stages of designing are: defining specifications of reusable IP, integration using standard coding methods, validation and coverage (functional, code coverage etc.). IP designing starts with proper functional and design description of IP. The next step comprises of code design, synthesis, and design for test. The second step takes only small amount of total time of IP design. The most important and third step of IP design that is verification of design takes up to 50 percent of the time. For reusable purpose it is very important to get error free IP. Ideally the goal of verification is to achieve 100 percent code coverage and close to 100 percent functional coverage.

2.1.5 SoC platform

As SoCs complexity and design cost increases day-by-day, it is very important to incorporate programmability within SoC to provide for reuse at the chip level. This programmability can be either hardware programmability (obtained by programmable logic cores) or software programmability (embedded processor). The key feature of programmable SoC designs is to provide flexible hardware and/or software infrastructure that is programmable fabric. Reusable IPs although speed up the soc integration, limits the productivity gains. Since the integration of SoC can be time consuming even after using predesigned and pretested IPs. Higher level of abstraction is still needed. That is an architecture that can serve many customers. The configurable architecture from which can be developed many other SoCs. For this, the platform based design was introduced so that new designs can be developed from a base platform to keep costs in check. In short platform is an abstraction level comprising of a number of features of lower level. Designer productivity is enhanced because many high level and low-level requirements are already present in the platform, also all tools and flows are already developed to fasten the design of new Soc. An SoC platform generally comprises of hardware IP, software IP, programmable IP, standard bus architecture and communication networks, CAD flows for hardware/software co-design, validation tools, design derivative creation tools and dedicated hardware for system prototyping.

2.1.6 SoC design flow

As shown in Figure 2.3, the SoC design ow starts with the design specification. SoC design specifies all the needs of end user. The architectural specifications along with the methodology to be used are finalized. Architectural validation is done. IPs are chosen from a huge IP data base according to the specifications needed. Following factors are considered while selecting an IP:

1. Performance, Power, cost and area
2. Hard IP vs Soft IP
3. Environment used for development
4. Flexibility

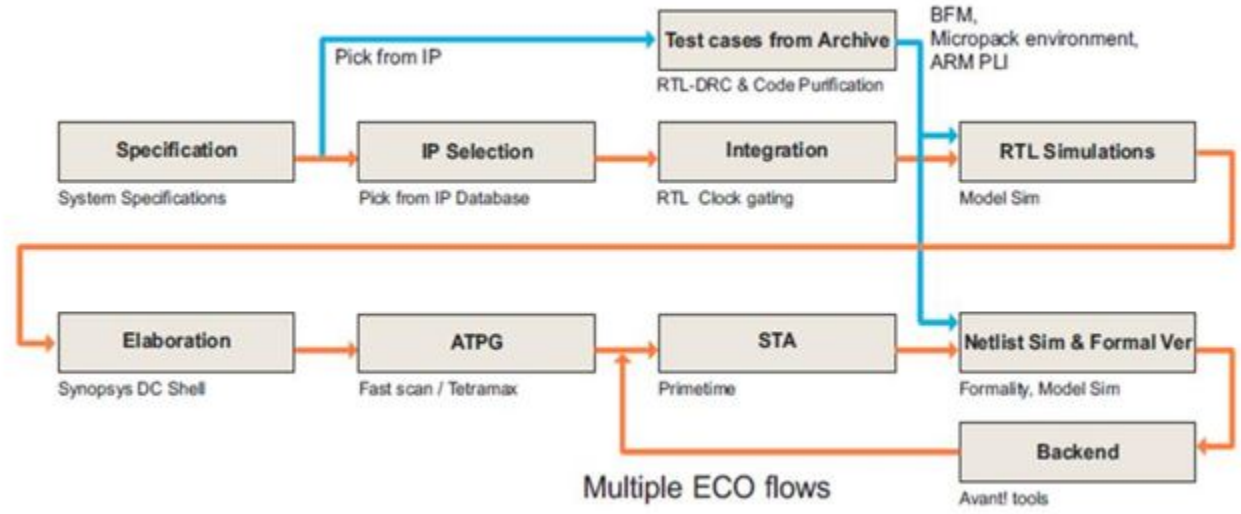


Figure 2.3: SoC design flow

5. Ability to operate with other IPs

6. System software available

7. Technology library

IPs are first individually verified before integration. Different IPs are integrated together that is they are connected together and to the system interfaces. RTL simulation is done to eliminate any RTL related errors at the preliminary stage. Simulation is the process to create a model or an abstract depiction of a system to recognize and understand system controlling factors. Simulation helps in predicting the future behavior of the system. Simulation is independent of technology and checks for any syntax errors. Elaboration stage performs compilation and gate level netlist is generated. Elaboration process is technology dependent and create netlist of the technology generic cells. Elaboration process creates hierarchy of design and instantiates all the submodules called within a module. It resolves hierarchy. ATPG (automatic test pattern generation) is used to generate random test patterns, to perform functional verification. Static Timing Analysis method is used to verify the timing performance of system. It checks for all possible paths between inputs and outputs for timing violations. Formal verification is used to verify certain algorithms of the system. It checks whether a design satisfies properties.

2.2 Magillem

Magillem, a major contributing member of the Spirit/Accelera Consortium, offers a comprehensive front-end EDA environment (also called Magillem) for IP packaging, concurrent platform integration, netlist generation, flow execution and register management based on IP-XACT (IEEE 1685-2009 and previous versions).Magillem is strictly based on IP-XACT standard with no proprietary extensions, which enables the user to remain tool and vendor independent.

The biggest advantage of using IP-XACT based register specification is the fact that when files like register bank RTL code, C headers and register documentation are generated from the same source there are no functional differences between the files and errors are minimized as long as the specification is correct. Checkers will catch the most common errors in register specification preventing issues like overlapping bit fields or overlapping address spaces of different IPs.

2.3 IP-XACT

Due to the lack of standardization of interfaces and concerns about configurability and quality, the design reuse has not provided the anticipated SoC design flow advances. There is also a need for regulated way of controlling the integration flow, automation and verification quality. The IP-XACT standard has been designed to address these issues by providing a tool-independent, standardized data exchange format to be used in flow automation and verification. It enables development of automated tooling by EDA vendors and has proven to be useful solution for IP reutilization.The IP-XACT standard uses meta-data to describe IP in a design language and tool neutral way. The standard aims to aid the delivery of compatible IP descriptions from multiple IP vendors, to improve the importing and exporting of complex IP to, from and between EDA tools and to improve the provision of EDA vendor-neutral IP creation and configuration scripts.The standard speeds up the software development and makes it possible to start it earlier in the design flow, thus providing a time-to-market advantage.

In SPIRIT, this an open industry-driven initiative, which develops a standard format for all kinds of data related to IP integration such as hardware constraints, debugging information and register descriptions.

It is an XML format and is an IEEE standard which has been developed by Accellera that describes and defines hardware and software components and their corresponding designs. Created by the SPIRIT Consortium, it is a standard which helps to enable automated configuration and integration explicitly through tools.

The objectives of the standard are:

- To ensure delivery of compatible component descriptions from multiple component vendors,
- To enable exchanging complex component libraries between electronic design automation (EDA) tools for SoC design (design environments),
- To describe configurable components using metadata, and
- To enable the provision of EDA vendor-neutral scripts for component creation and configuration (generators, configurators)

It is approved as IEEE 1685-2009 on 9th December, 2009 and got published on 18th February, 2010 As the SPIRIT consortium is composed of major EDA companies, there is developing tool support for the metadata specific format. For example, the tool Denali Blueprint supports IP-XACT to read register descriptions. A metadata schema is provided by the IP-XACT specification for the description of IPs integrated in SoC thus enabling it to become compatible with tool automated techniques for integration, and an API for tool access to this schema. Tools using this standard for implementation would be able to automatically configure, integrate, interpret and manipulate IP unit blocks that are delivered with metadata that complies with rules to the proposed IP description from metadata, and a standard method is provided by IP-XACT APIs for linking multiple tools through a single exchange format specified in metadata. An IP-XACT enabled environment is created from multiple vendors for automatic integration of tools and IPs.

Once all the IP meta-data is added to its IP-XACT component description, checkers can be used to verify the IP-XACT data. Checkers provided by an IP-XACT design environment make sure that components, designs and other IP-XACT XML descriptions are correct according to the SCR in terms of syntax, mandatory properties, content and the structure of the schema. Tools may have their own checking rules as well. Verified components, bus definitions and abstraction definitions

can be added to an IP-XACT catalog file. All IP-XACT components and bus definitions can then be imported into a new IP-XACT project or design environment by using the catalog file.

2.4 XML

Extensible Markup Language (XML) is a language that defines a set of predetermined rules for structuring and encoding documents in a specific format that is both machine and human readable. Several schema exist in the definition of XML-based languages to aid the processing of XML data.

Understanding IP-XACT requires basic knowledge of the XML language. XML is used to structure, store and transport data in plain text format. The XML files themselves do not do anything. They just present data in a software and hardware independent way which makes them ideal for transporting data between incompatible systems or preserving data while upgrading to new systems. Hardware or software platform changes generally require large amounts of data to be converted and the incompatible data is often lost. XML solves these problems because the only required feature for applications handling XML files is text processing.

XML files consist of data elements wrapped in tags. The functional meaning of the tags depends on the nature of the application. The language itself has no predefined tags. The tags as well as their structure is completely determined by the user. They can be arranged according to an XML schema which is used to define a list legal tags and how they should be structured. It specifies which elements and attributes can appear in an XML file, which elements are child elements, the number and order of child elements and whether an element is empty or can include text. An XML schema also defines the data types of elements and attributes as well as their default and fixed values.

2.5 SystemRDL

In 2005, Denali released a tool called Blueprint that used a format called Register Description Language (RDL) for describing registers. SystemRDL was then adopted by the SPIRIT and released as SystemRDL 1.0 in May 2009. Supported by the SPIRIT Consortium, the SystemRDL language was specifically designed to describe and implement a wide assortment of control status

registers. Using SystemRDL, developers can automatically synchronize and generate high level register views efficiently for specification of hardware components, hardware design, software development, verification, and documentation.

SystemRDL is the only open source text based descriptive language that focuses exclusively on registers. SystemRDL1.0 had some limitations which are now being worked on by a group under the auspices of Accellera. SystemRDL2.0 will have support for verification based properties like Constraints, Coverage, and HDL_PATH etc.

In contrast to the current IP-XACT definition, SystemRDL concentrates on register specifications only. It provides many constructs that have already a corresponding counterpart in IP-XACT, such as registers, fields and complete address maps. Additionally, SystemRDL has a much wider range of constructs to model functional interdependencies between registers, enabling the generation of more detailed and complex models of registers. On the other hand, due to its rich language, the processing overhead for SystemRDL descriptions is higher than for the common XML format.

2.5.1 SystemRDL Components

Component is a basic building block in SystemRDL that acts as a container for information. This is similar to a struct or class in programming languages as shown in figure 2.1.

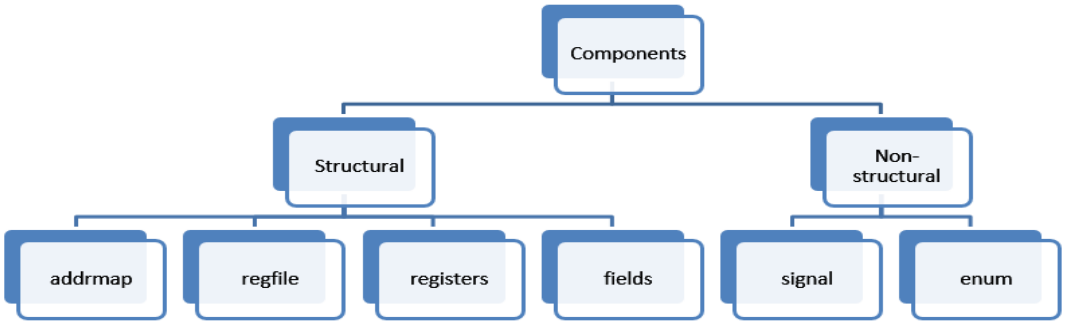


Figure 2.4: Classification of SystemRDL Components

SystemRDL- Structural Component

- A field is the most basic component object and serves as an abstraction of hardware storage elements
- A reg (register) s a set of one or more fields which are accessible by software at a particular address
- A regfile (register file) is a grouping of register files and can be organized hierarchically. A regfile may not instantiate other regfiles.
- An addrmap (address map) defines the organization of the registers, register files, and address maps into a software addressable space. Address maps can be organized hierarchically.
- A reg has to have at least one field
- An addrmap should contain either only other addrmap or only reg/regfile

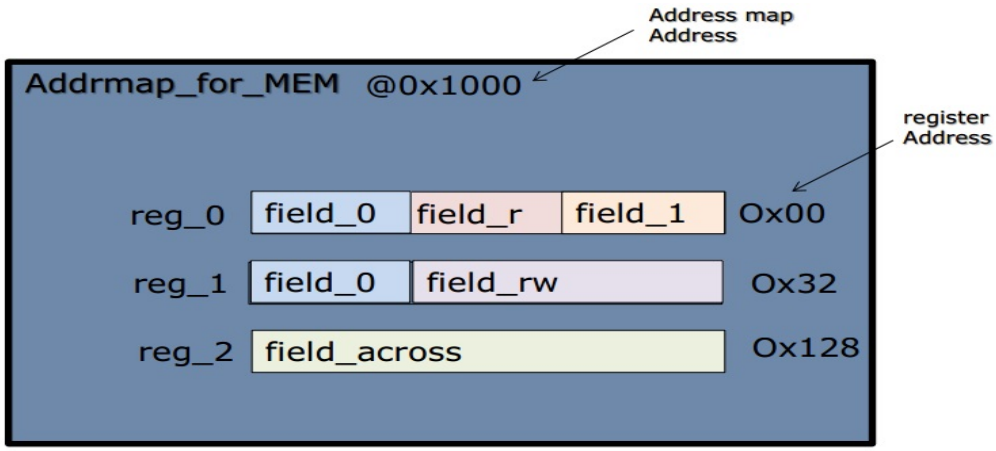


Figure 2.5: Registers containing fields within an addressmap

SystemRDL-Non-Structural Component

- A signal is used to define and instantiate wires. They create named external ports and can connect certain internal component design property to the external world
- An enum (enumerates) describes a set of that provide mnemonic names for field values.

2.5.2 SystemRDL-Properties

Property is a characteristic, attribute, or trait of a component in SystemRDL. They are used to store information.

- Properties are only valid for specific component type(s)
- Output generators use properties to add information the generated collaterals such as HTML or XML

Native property: There are a set of natively supported properties in SystemRDL which include desc (description), name, etc.

- These labels are always lower-case

User-defined properties (UDP): Properties that aren't supported natively are encoded in User-Defined Properties (UDPs) in SystemRDL.

- UDPs allow additional user-defined properties to be described. E.g. AccessType

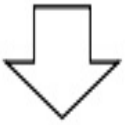
A property outside of the base set of properties defined in SystemRDL. Intel SoCs have standardized on specific UDPs applied at the addrmap, reg, or field component within RDL.

Attribute applied to any SystemRDL component at the field/reg/addrmap level. Property/attribute values are formatted in generated output code.Properties may be native to the SystemRDL format or may be part of a User Defined Property (UDP) list.

2.5.3 SystemRDL-Text Substitution with Perl

Any code snippet beginning with <% and terminated by % >shall be evaluated as a Perl scalar value which shall replace the snippet in a preprocessing phase when the RDL is compiled. The following example shows the use of the SystemRDL preprocessor:

```
reg myReg {
        <% for( $i = 0; $i < 3; $i++ ) { %>
        myField data<%=$i%> [<%=$i*2+1%>:<%=$i*2%>];
        <% } %>
};
```



```
reg myReg {
        myField data0 [1:0];
        myField data1 [3:2];
        myField data2 [5:4];
};
```

Figure 2.6: Example of text substitution in Perl

2.5.4 SystemRDL- Basic Register Example

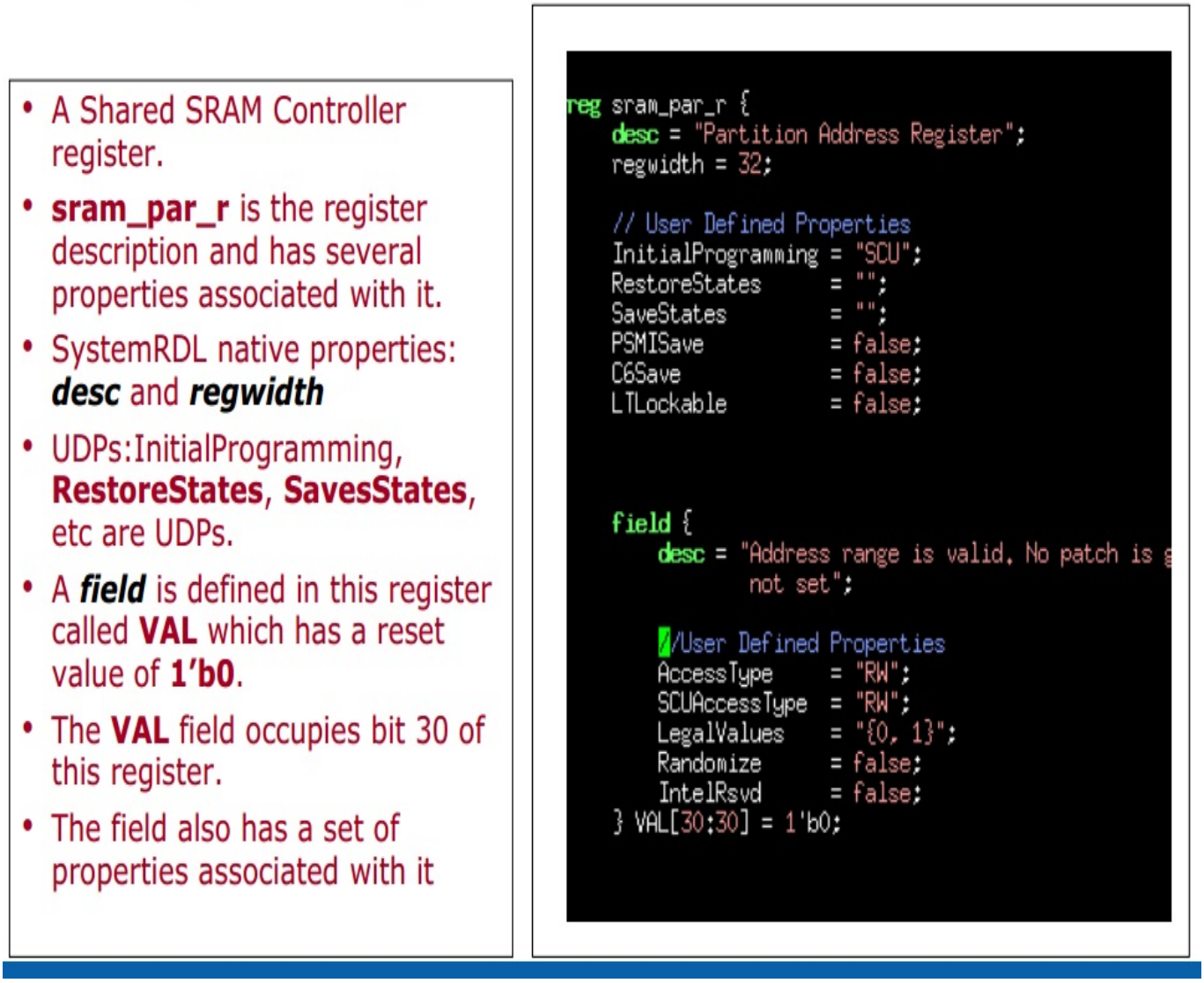


Figure 2.7: Basic register example

2.6 IP Block Config Register Definition

Each IP block should be independently represented as a collection of registers. Each group of registers (or sub-map) can be defined as an addrmap. Each addrmap at the IP level represents a bank of registers assigned to a named system space (MEM, IO, CFG, and MSG). At the system level, the IP integrator can assign a common address to a particular named register space.

Each register in a particular block must be represented as a reg definition. One way to define all registers in an IP block independent of their address map spaces is to create one or more .rdlh files to be included inside an addressmap.

Instead of using the native “hw” and “sw” properties in SystemRDL, the expanded UDP “AccessType” should be used to specify the hardware/software access attribute for a register.

```
reg rega {
  name = "rega";
  desc = "this is the A register for the unit";
  regwidth =16;

  field {
    name = "fieldXYZ";
    desc = "XYZ control field";
    AccessType = "RW";
  } XYZ [7:0] = 8'h0;
  field {
    name = "fieldLMN";
    desc = "LMN control field";
    AccessType = "RW/O";
  } LMN [15:8] = 8'h0;
};
```

Figure 2.8: Register Hierarchy

Chapter 3

Virtuous Cycle of Register Standardization

Virtuous cycle of register standardization defines the process starting from register specification template to consumer feedback incorporation for all program through a systematic assessment and review process for any future inclusion on the requirements. Overall process has been divided in to 5 major section with specific requirements at each stage and guidelines to be followed by producer/consumers.

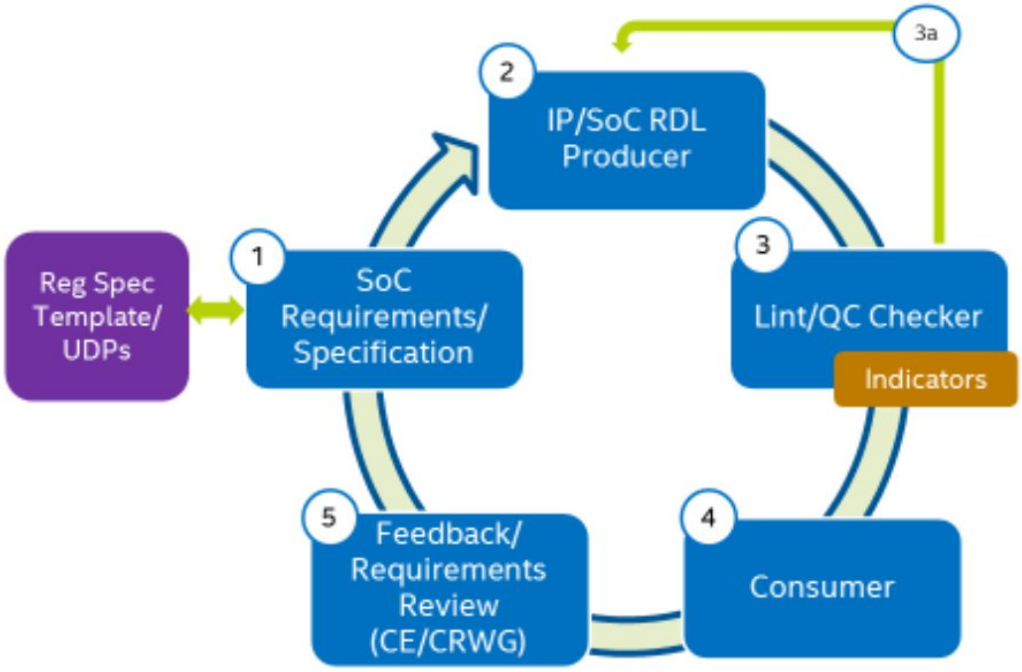


Figure 3.1: Virtuous cycle of register standardization

1. Putting together SoC specification based on standard RDL template with UDP owners
2. IP/SoC Producer to generate RDL
3. RDL goes through LINT/QC checker (3a-Feedback for QC Error Fix)
4. Release to consumer for review usage
5. Feedback/additional requirements, review with content expert and CRWG for approval and update

3.1 Elements of Virtuous Cycle

3.1.1 SoC Requirements and Specification

Each project requirements and delta should be captured to map against the template with required UDPs to develop SoC Spec which should be used for developing new register definition. All approved UDPs with clear owner should be assigned to define scope of work for producer to develop new registers for any project. This servers as a starting baseline for different IP/SoC RDL producers to add/modify new or existing registers.

3.1.2 IP/SoC RDL Producer

IP/SoC RDL producer should follow the required guidelines for defining any new register or changes required for existing registers. Any addition/change should be complaint with RDL rule check to avoid any feedback and turn around after QC error check. IP owners should ensure that none of the rules are violated before releasing it to SOC team. SOC team should follow the same for SOC specific IPs Registers. All IP and/or SOC specific blocks should be released along with POR QC error check qualification and reports.

3.1.3 Lint/QC Checker

As a part of release process tool based QC check should be run to ensure none of the guidelines and quality checks have been violated and final report should be delivered along with RTL milestone.

In case of any failure, required RDL update should be sent back to IP/SoC RDL producers before sharing with respective consumers. Any waivers based on the approval must be documented as a part of release notes. Standard indicator as defined will be published periodically for each release.

3.1.4 Consumer

Final version of QC checked RDL will be released to each consumer as required to develop their collateral for downstream consumers. Any violation to the process so far in terms of quality should be flagged by consumers to the review committee.

3.1.5 Feedback/Requirements review (CRWG)

Any new requirements in terms of UDPs or inappropriateness of quality check should be flagged by the consumers which should be assessed and reviewed by the standardization committee before inclusion to register spec template. CRWG will review all feedback to ensure that only must have and appropriate content is added to avoid any redundant UDPs or definitions included to Spec along with any correction action or update required for QC error check. This standardization will ensure standard processes to be followed by all producer/consumer team to ensure quality and deliverables. Any violation will be flagged through the indicators at each milestone and will be reviewed by the CRWG.

3.2 Tool Flow

The tool used here in INTEL is a DTS (Design and Technology Solutions) tool that is used to process the SystemRDL files and create output collateral. It includes an EDA vendor SystemRDL parser/engine.

- **Distribution/Availability:**The tool flow is integrated into the design environment and is a part of the design tool chain and has a quarterly (or more) scheduled releases
- **Interoperability:**Although the tool produces many files that are used in downstream design flows, the interoperable format passed from one project to another or from IP to SoC is SystemRDL and not the output files

- **SystemRDL is the golden (single source) data:**The version control repository tags the SystemRDL and RTL files at the same time so that accurate snapshots can be traced back to specific RTL versions

The collaterals generated from SystemRDL files are

- XML: this is a SPIRIT/Accellera defined format
- HTML: a viewable/searchable file
- FirmwareC-header: for firmware
- CRIF: Control Register Interchange Format (XML)
- CReg: Control Register format (XML)
- CSPEC: word xml used for documentation
- OVM: RAL compliant validation code
- RTL: System Verilog Implementation of the registers
- Fuse: FUSE manager validation code + doc + implementation files
- DFX: TAP manager validation code + doc + implementation files
- Scan: Scan manager validation code + doc + implementation files

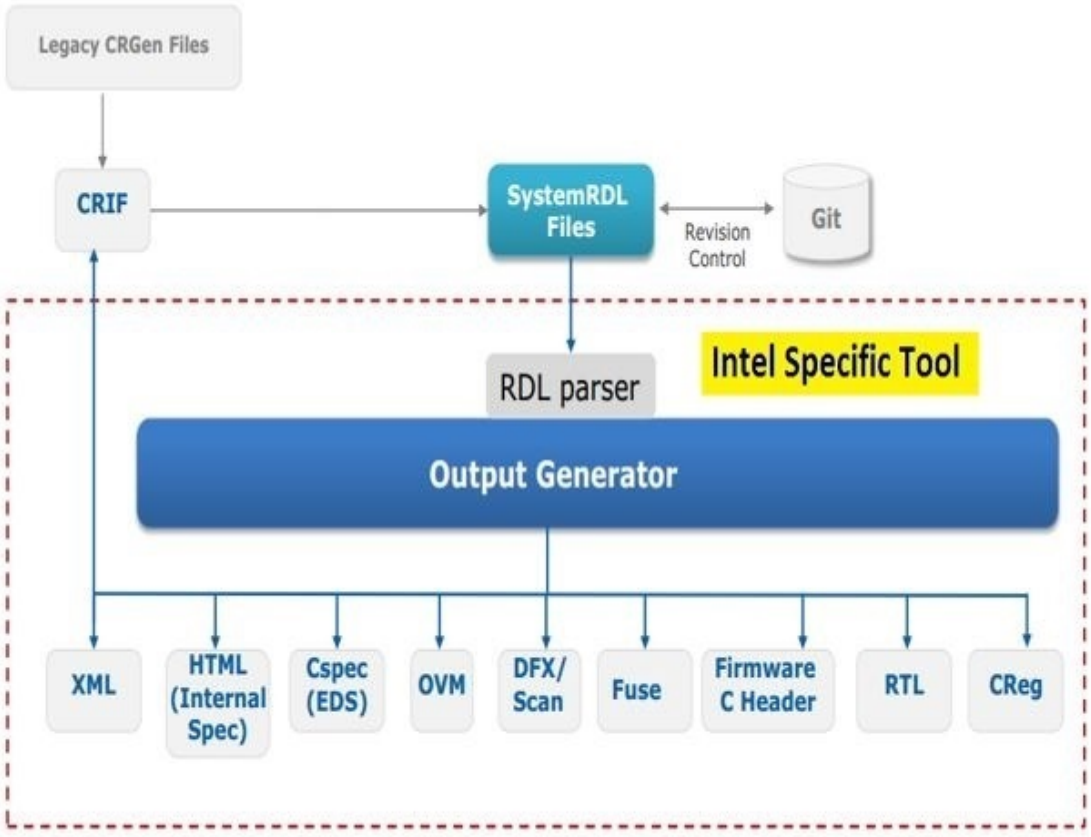


Figure 3.2: Tool flow

3.3 CRIF-Control Register Interchange Format

Using the –crif option will generate a Control Register Interchange Format (CRIF) style XML tree representing the addrmap hierarchy and all associated properties in the data model extracted from the RDL. The XML structure is based on IP-XACT but includes additional attributes for content, register files, collections, MSR, and registers/fields.

CRIF is an internal format to allow interchangeable register formats to import into CRGen and CRWebViewer.

Within Intel, there are three major methodologies for processing control register (CR) data. Tool1 and its associated proprietary input format is used within CPU projects. SOC-DA/SIP/chipsets use Tool2 (described here) which sits on Cadence's Blueprint tool using SystemRDL input. GT uses Tool3 and associated tools. In addition, some projects are still using legacy old tool.

While these solutions work well within their environments, there are several ways that this division is limiting. These limits include obstacles to IP sharing and increased burden on post-design groups. As Intel shares more and more IP blocks, CR data is crossing design group boundaries. CPUs are absorbing more blocks from both GT and SEG which need to be properly integrated into the larger product. Additionally, some chipset products may need to absorb one or more CPU cores. Post-design activities include BIOS, system validation (SV), documentation, and others.

Long term, Intel would like to converge on a single set of tools and CR specification format, however, if this change takes place, it will take time. In the interim, Intel groups have agreed to converge on a common output file format that can be used to communicate CR data. Such a format will continue to be useful even after input and tool convergence because it is designed for ease of accessibility to machine parsers. These files are intended to be downstream only and no consideration has been given to their human edit ability. While a register may contain an attribute telling of its instanced origin, each instance will be listed as a separate register.

3.3.1 Basic Structure

Each level of RDL hierarchy is represented by indentation starting with the top-level addrmap and properties, followed by the instances of sub maps, registers, regfiles, and register fields.

```
<crif>
  <registerFile>
    <name>ABC</name>
    <BaseAddress>32'h0</BaseAddress>
    <type>CFG</type>
    <bus>0</bus>
    <device>1</device>
    <function>0</function>
    <register>
      <field>
        <name>DEF</name>
        <designName>DFD_XYZ</designName>
        <access>RW</access>
      </field>
    </register>
  </registerFile>
</crif>
```

Figure 3.3: Crif syntax

3.3.2 Generator Outputs

The -minimize_crif command-line option removes description, ValRTLSignal and other non obligatory attributes from generated CRIF output, significantly decreasing its size. This option is recommended for very large designs, which CRIF becomes too large to use.

The -crif_line_notation command-line option replaces "[]" brackets with "_" notation in arrayed registers in generated CRIF output.

The -crif_hide_default_udps command-line option prevents properties with value equal to their defined default from being printed to the generated CRIF output. Only properties with different from default value will enter the CRIF output, significantly decreasing its size.

The -format_numbers command-line option will instruct tool to use unique format for hex numbers in generated CRIF output.

3.3.3 How is CRIF different from IP-XACT?

IP-XACT is a standard format designed for the purposes of IP integration. CRIF format follows the IP-XACT format for the attributes that it specifies. CRIF takes this further with many attributes not mentioned in the IP-XACT format. These additional attributes include: an expanded list of possible access types, MSR vales, software save, bit encodings, and others.

It's the adoption of and convergence on important attributes list those listed above that make a format truly and interoperable. This is why the files are called CRIF.

Each flow provides register data in CRIF file of its own. Such file can only contain data in context of given flow or IP. For example CRIF file of any IP block can only specify relative offsets of the registers and not the full addresses in final product. Same holds for RTL location of the register: it can only be specified from IP top level and down.

CRIF merger utility consumes these CRIF files with additional data provided by integration team, and generated project CRIF file, with exact mapping of register groups to memory spaces, full addresses and RTL paths, etc. This file is consumed in post-silicon environment.

3.3.4 Register File

At a high level, the intent of register files is to describe address spaces or domains. There are some instances where two register groups contribute to the same space, so this division should not be seen as absolute. An example of this could be where several IP blocks add to the same address space from different register groups.

Buses are not included as a hierarchy. They are not as meaningful in all cases and some designs can straddle multiple buses. EDRAM is an example of this.

Register files represent architectural organization rather than physical. Most consumers of register collateral are not interested in the physical aspects. Defining what a register group means is best done by listing the naming to be used for current CPU registers.

3.3.5 Unknown Address space

There will be many cases where the IP provider has no knowledge of the target address space.

In these cases, the IP provider will need to come up with a good unique name. There is no requirement that the register group name include B*_D*_F*. When these IP blocks are absorbed into the target project, address space attributes will be merged in as appropriate. The name would only exist in one place in the file anyway, so that is something that could be changed, if necessary.

For this to work fully it likely might be required that a "Pointer" be allowed to be created in "descriptions" so that IP providers can specify a different register inside the reg_group (or a related reg_group). Then when descriptions are resolved the "pointer" in the descriptions can be resolved to the full name.

3.3.6 Naming uniqueness

To avoid ambiguity in design, element names must be unique within each level. Field names must be unique within a register. Uniqueness of bit names across registers is not required. Register names must be unique within a registerFile. RegisterFile names must also be unique. This is the only place where some amount of coordination will be necessary across Intel design groups.

If instances are found where two elements share a common name, tools will merge them together. This will enable one IP block to augment registers or bits of another block.

On a related note, many current registers contain b/d/f in their names, over time these names will be modified to remove that redundant information. More immediately, CRGen will modify the names to put this change into effect immediately. For activities that require the name as seen by design, the attribute rtl_name will contain the original. Because the original rtl_name will be included, correlating between source data and CRIF files should be straight forward.

3.3.7 Register Address

Register files will occupy continuous parts of the relevant address space. Owners of the register files will provide address offsets of the registers; the integration team will provide a base address of the register file in relevant space. Register final address is a sum of them.

3.3.8 Internal vs External registers

Some registers are communicated to groups outside of Intel. Many are not. The CRIF file will need to be different depending on the intended audience. The most obvious difference is the omission of internal registers from the externally targeted file but there are others. For example, the naming of registers can be and will be different. Internal names can contain contextual hints useful to designers. They can reflect micro-architectural information. They can be named to be consistent with shadowed registers which are not visible to the outside.

Whether or not a register is classified to be external will change over the life of the project. Because of this, internal use files need to retain usage or internal names as their primary identifiers. External use files will have registers named by their "external_name" attributes.

In some projects, it may not be necessary to generate these two file variations; the documentation team can do the appropriate filtering and modifications. For other projects, these duties will be owned more by design itself. In particular, there are cases where CRIF files will be included in the collateral that is communicated outside of Intel. This is already the case with the old style cregs.xml files.

External name

Externally visible registers have two names: the external name and the internal name. For files intended for external consumption, only the external name will be given. The external names must be unique within their register group. In internally targeted files, the name given to a register will often be different (in the CPU context, generally ccb_cr register name). The external name will be given as an additional register attribute.

External visibility

Even if a file is written for the purposes of external consumption, it is often necessary to perform additional scrubbing to exclude/include relevant registers. Documentation generation is a notable example of this. Documentation groups will receive a specification containing all external registers and they will generate multiple versions of register documents. To meet this need, all external registers must include a protection_level tag with one of the values “red”, “orange”, or “green”. Some design groups may elect not to go this round, choosing instead to generate multiple versions, one for each protection level.

3.3.9 Access

While there are many values that access type can take, there are really only a handful of base/primary access types and then many versions of those created by applying an access type modifier. Here is info on those modifiers:

The ”S” reset sticky modifier. Exists in any access type that would be identical to another access type if the ”S” just before the _ (or at the end if no _ exists) were removed. This means that the field is only reset on a powergood reset.

The following modifiers exist after a _ (there can be multiple):

*: Variant, meaning hardware can update the value at a time other than reset.

*: Can only be written once after which it is locked (until reset).

*: Writes to the field are locked by the value of a different field (same or diff reg).

*: This field is the lock key to other fields.

*: Firmware can write to any value even if the primary access type restricts writes.

*- Intel Confidential. Cannot be revealed.

3.4 Advanced Features

3.4.1 PnP

What is not included in RDL is the inter-knob behavior, i.e. how knob1 affects knob2. This information is in the HAS which has higher-level explanations of all the knobs and how they affect the SoC in general.

Power and Performance (PnP) information allows DE, VE, and software developers of the optimum settings for a particular register bit (i.e. knob) that influences Power and Performance of an IP block.

UDP's: 5 UDPs for PnP

Property	Type	Default	Component	Description
ImpactsPwrPerf	Boolean	False	Register, Field	This tells if a register or field is a Power or Performance knob and the ValueFor* UDPs have to be filled out correctly.
ValueForPerf	Number	0	Field	ValueForPerf is the setting for highest performance.
ValueForPower	Number	0	Field	ValueForPower is the setting for lowest power consumption.
ValueForPwrPerf	Number	0	Field	ValueForPowerPerf is the optimized setting that balances power and performance. If there is an issue with determining this number then this should be biased towards being power efficient.
ValueForPwrPerf_Range	String	Blank	Field	ValueForPwrPerf_Range is a string that is used for differences in PnP settings.

Figure 3.4: UDPs for PnP

A field that impacts PnP is defined. The values that are expected to be chosen for the different optimizations are given.

- The range is 30 to 60
- 30 is optimized for power

```
reg pnpknob {
  field {
    name = "Knob 1";
    desc = "This knob affect power and performance.";
    AccessType = "RW";

    // PnP Settings
    ImpactsPwrPerf = True;
    ValueForPower = 30;
    ValueForPerf = 60;
    ValueForPwrPerf = 40;

    // If required for different ValueForPwrPerf
   settings
    ValueForPwrPerf_Range = "266MHz=34;
                             400MHz=36;";
  } knob1 [15:0] = 40;
};
```

Figure 3.5: PnP Example

- 60 is optimized for performance.
- 40 is a recommended default
- Specific values, semi-colon separated are given in ValueForPwrPerf_Range

3.4.2 HDL Signal Path

Adding HDL signal names into the RDL file assists validation in finding signal paths for forcing/querying for white box signal access.

- ValRTLSignal is a field or addrmap property that is used to define the RTL signal path.
- ValGLSSignal is a field or addrmap property to define the gate level/netlist signal path. (This may be a list of signals).

3.4.3 Lockable fields

- Field's value cannot change when the locking field is in locked state
- Requires LockKeyField and LockKeyValue properties to be specified in RDL
- Field in regA locks field in regB

Figure 3.5 gives the description of the lockable fields with the two registers regA and regB.

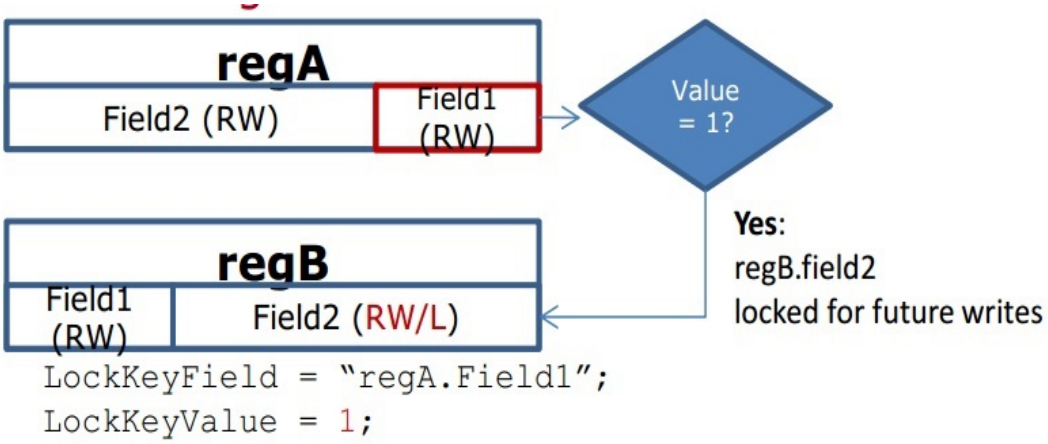


Figure 3.6: Lockable fields

3.5 SAI (Security Attributes of Initiator)

Security Access Control information is used to identify which agents have access to a particular register. Access control information is embedded into the RDL files with special UDPs.

3.5.1 SAI Register

Registers fall into two categories:

Policy Registers

Policy Role may be

- CP (write access to itself all policy registers in group)
- WAC (write access to functional registers in group)
- RAC (read access to functional registers in group)
- AC (read+write access to functional registers in group)

Functional Registers

- Legacy IPs with fixed (predefined) SAI access: IP controls which agents can read/write register

- Dynamic access (non-legacy): IP gets its register SAI properties from the SoC (which can override abc.pm)

The RTL Generator supports SAI (Security Attributes of Initiator) checking on a register-by-register basis to allow/disallow reads and writes. The user specifies a ”group” for each register via the PolicyGroup UDP. This is a user-defined UDP provided in the source RDL file. There may be multiple groups.Each group consists of control policy registers and slave registers (slave registers are just normal registers to be protected). If a register does not have a group, no SAI checks are performed upon accesses to that register.

All control policy registers are always readable; there are no SAI checks performed on reads to the control policy registers. The control policy registers contain a bit per SAI to determine if an incoming cycle with that SAI is allowed to access the register or not.

Chapter 4

Quality Checker

Using the –qualitychecker option will only perform a check of the RDL file (and included RDL files) for missing/incorrect property values. Quality checker of the tool for IPDS (IP development standard). IPDS is targeted at IP quality, but checking is also appropriate at the SOC level, when additional information is known.

The quality checker log file contains lacs of lines. To segregate individual errors and to generate the list of IP names from which errors are reported and its corresponding total counts, Perl scripts were developed with the help of which the flow of debugging became faster and easier.

4.1 Errors Debugged

4.1.1 Missing access type

Message

AccessType is not specified for field '%s'.

Description

Missing AccessType. The fields must include AccessType property.

Attributes

Name: Missing AccessType

Category: Rdl

Sub Category: Rdl general

Language: ALL

Severity: Fatal

4.1.2 Missing register name

Message

Name is not specified for register '%s'. The attribute 'name' is missing.

Description

Missing register name. All registers must include the name property representing the documentation name.

Attributes

Name: Missing Register Name

Category: Rdl

Sub Category: Rdl general

Language: ALL

Severity: Fatal

4.1.3 Address collisions

Message

The register '%s' has final address '%s' and width '%s', while the register '%s' has final address '%s' and width '%s', they are in the same space '%s' and occupy the same address.

Description

Two registers in the same space may not occupy the same address. If two registers are in MEM space with equivalent/equal BaseAddress (BAR) values, their addresses may not overlap. Similarly, CFG with identical BDF and MSG with the same portid.

Attributes

Name: Address Collisions

Category: Rdl

Sub Category: Rdl general

Language: ALL

Severity: Fatal

4.1.4 Invalid AccessType

Message

Invalid AccessType value '%s' for field '%s'.

Description

Invalid AccessType. The fields must include AccessType property.

Attributes

Name: Invalid AccessType

Category: Rdl

Sub Category: Rdl general

Language: ALL

Severity: Error

4.1.5 Invalid SB_Fid for space msg

Message

Invalid SB_Fid value '%s' for Space type MSG in addrmap '%s'.

Description

Invalid SB_Fid for Space MSG. Addrmap with Space MSG and Opcode CFG-SB/MEM-SB, must have a SB_Fid property. SB_Fid must be an 8-bit value and must equal the value of (device SHIFTLEFT 3) + function as listed in the sibling CFG space addrmap.

Attributes

Name: Invalid SB_Fid For Space Msg

Category: Rdl

Sub Category: Rdl space

Language: ALL

Severity: Error

4.1.6 Mem BaseAddress register is not defined

Message

BaseAddress '%s' of the addrmap '%s' references an undefined register '%s'.

Description

Invalid BaseAddress for Space MEM, BaseAddress references an undefined register. Mem BaseAddress may be in one of the following forms: a constant number

Attributes

Name: Mem BaseAddress Register is Not Defined

Category: Rdl

Sub Category: Rdl space

Language: ALL

Severity: Error

4.1.7 Policy group with multiple registers for the same role

Message

The Security Policy Group '%s' has the register '%s' and the register '%s' with the same Security_PolicyRole '%s'.

Description

Policy group must not have more than one register for the same Security_PolicyRole.

Attributes

Name: Policy Group with Multiple Registers For Same Role

Category: Rdl

Sub Category: Rdl Sai

Language: ALL

Severity: Fatal

4.1.8 Invalid Description

Message

Invalid Description '%s' for %s '%s', desc attribute must not include invalid XML control characters and must not include the following: TBD, tbd, fixme or FIXME.

Description

Invalid desc attribute. Desc attribute must not include invalid XML control characters and must not include the following: TBD, tbd, fixme or FIXME.

Attributes

Name: Invalid Description

Category: Rdl

Sub Category: Rdl general

Language: ALL

Severity: Error

4.1.9 Missing LockKey Field

Message

LockKeyField is not specified for field '%s' with AccessType '%s'.

Description

Missing LockKeyField for lockable field. All fields with a /L AccessType values must have LockKeyField property specifies the name of the trigger/key field that controls the state of the lockable field.

Attributes

Name: Missing LockKeyField

Category: Rdl

Sub Category: Rdl general

Language: ALL

Severity: Error

DIE								
ERROR	CRIF_WW27.5	PERCENTAGE	CRIF_05P_OS	PERCENTAGE	CRIF_WW30.4	PERCENTAGE	CRIF_WW34.1	PERCENTAGE
(Missing Access Type)	14	0.011439404	67	0.055857073	14	0.011456253	14	0.012726231
(Missing Register Name)	2702	2.207804942	2722	2.269297785	2162	1.769172858	2178	1.979838013
(Address Collisions)	380	0.310498104	344	0.286788552	344	0.281496514	344	0.312701688
(Invalid name)	705	0.576055694	709	0.591084544	705	0.576904193	705	0.640856657
(Missing ValRTLSignal)	86824	70.94391424	81346	67.81715562	86834	71.05659389	96067	87.32649147
(Invalid Register Name)	97419	79.60109165	95647	79.73972272	97385	79.6905175	85120	77.37548746
(Missing description)	78534	64.17015296	78419	65.37695187	78535	64.26549049	67417	61.28316774
(Register does not have a Security policy group)	83735	68.41989149	80135	66.80755988	83703	68.49448463	71241	64.75924697
TOTAL VIOLATIONS	378900		361410		376936		349894	
TOTAL REGISTERS	122384		119949		122204		110009	

Figure 4.1: ERROR counts with crif release

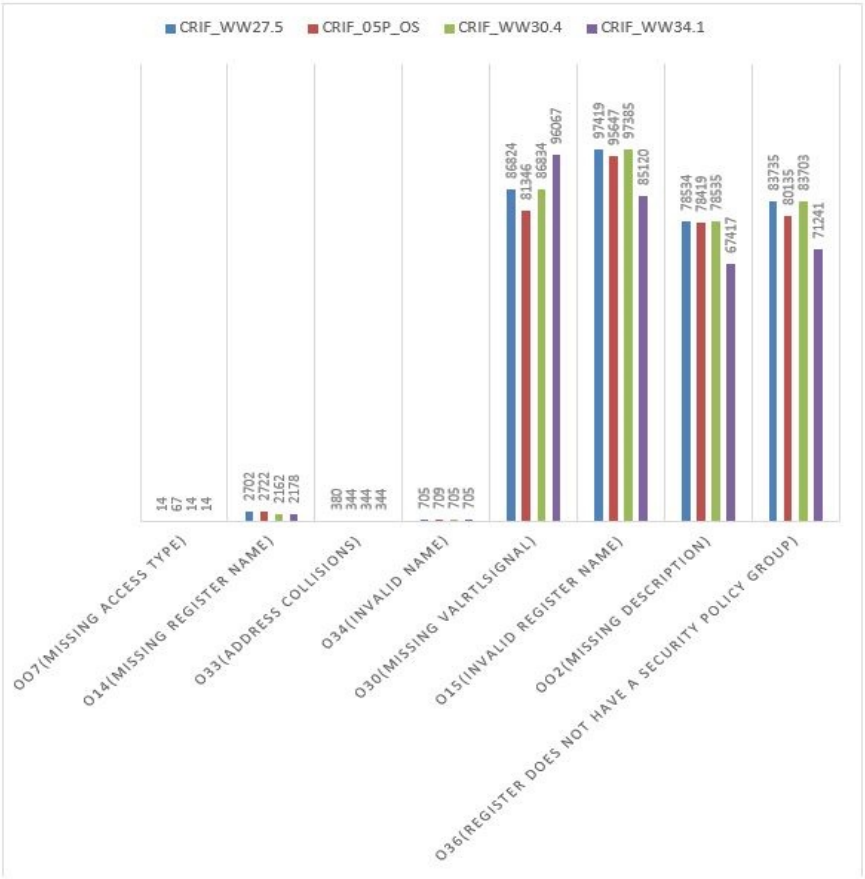


Figure 4.2: ERROR Comparison Chart for different CRIF release

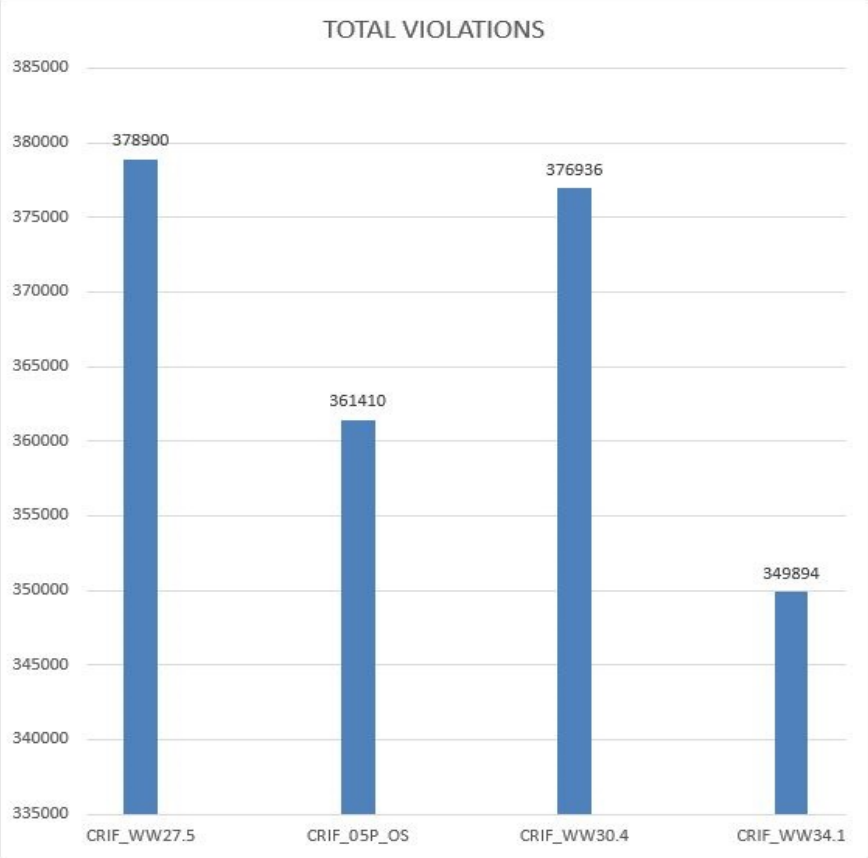


Figure 4.3: Total Violations

4.2 Lint Checks

When the lint checks are applied over the RTL by running a test, if the test is successful in running, a violation file in xml format will be generated. It can be imported by the lint tool for further analysis. A sample of graphical representation of different categories of violations shown in the lint tool at the subsystem level is as shown in Fig.

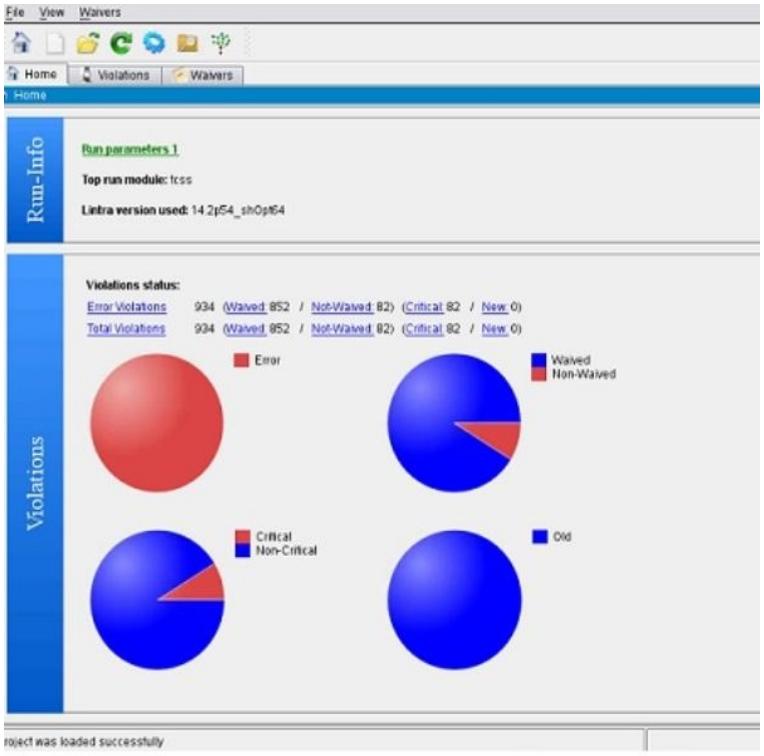


Figure 4.4: Lint Checks

Once the list of violations is obtained, the design engineer can decide if a violation can be waived. The waivers can be created and/or edited from the GUI, or through manual editing of the waiver file, which is in XML format. If a violation is to be fixed, one has an option to select the particular violation and editing the HDL file directly to fix the violations.

4.3 RDL paranoia Checks

Paranoia involves intense anxious or fearful feelings and thoughts often related to failure.QC covers mostly all the checks but still there are some which are not covered.So before tape in, some extra checks are performed to ensure quality RDLs get delivered to downstream consumers.

- Write Once fields in Save Restore

Field '%s' has write once related access type , that is problematic. It does not support save/restore. (SOixSave or SpecialSaveRestore or RestoreOrder or RestoreGroup or S0iXSaveRestore). If the field has *W/O* accesstype,the register should not have S/R attributes because "write-once" register will not allow us to do a restore.

- Waived Quality checker rules

In the IP drop process some IPs are waiving qualitychecker errors, and it might cause fatal errors. Need to check if there are quality checker rules that might have been waived in the TI process. Each IP owner will confirm that he run qualitychecker on his IP, and attach the waiver file to the ticket. Each waiver file can contain only rules that are not in the list of 'unwaived" rules that was written by the integration to SoC owner.

- Sync security files

There should be proper sync between local security and central security file.Every IP local security should point to central security file.

- Non-std attribute SoiXSaveRestore

Due to legacy issues, use of this attributed is prohibited. There should not be instance of this in the code.

- ExtendedID-no SB_Fid

The ExtendedID property can be used to indicate that a register exists in multiple CPU core/thread/module scopes without redundantly declaring the register in multiple addrmap. We have only one request

fid that can point to ExtendedID or to SB_Fid, not to both of them.In the packet, there is only one place to specify this.Now if we have extended id, that id will be used in the packet instead of SB_Fid. If we have both, it cannot be sent over the same band.

- Define Save/Restore values

There should be save restore definitions accurately present for all the IPs.There are different Save Restore attributes which clearly indicates whether the register is saved/restored.

- IP HW/SW priority

Updates HW or SW priority when a register supports both software update and hardware update . Only applies to registers with /* accesstypes. Architect should give the specific priorities spec related to all such Access Types.

- Volatile /* accesstype should come with RTLUseStar = FLOP

RTLUseStar UDP is used to switch on and off extra hardware. When using SharedRegs for some RO access_types, we implemented RTLUseStar = ”FLOP” which will instantiate an optional flop meant to hold the state of the register across a power reset event.

Chapter 5

CREST

CREST is Intel's Converged REgister Specification Test. It is a SystemVerilog/OVM-based test sequence that leverages RAL (Register Abstraction Layer) to perform basic control register validation, including

- register reset testing,
- field attribute testing,
- lock testing,
- shadow register testing, and
- Chassis-based access control (i.e., SAI) testing.

The algorithms behind each test, as well as known test limitations are documented here. Each algorithm queries the RAL to discover registers to be tested and to understand the registers' attributes and security properties. The result is an IP-agnostic test sequence that performs IP-specific register testing.

OVM:Open Verification Methodology

An open, interoperable SystemVerilog verification methodology. The OVM provides a library of base classes that allow users to create modular, reusable verification environments in which components talk to each other via standard transaction-level modeling (TLM) interfaces.

Register Abstraction Layer (RAL)

Configuration/Status Registers (CSRs) make up a significant portion of the verification environment for an SoC and their configuration values affect behavior of many other RTL and verification components. The quantity and complexity of these registers as well as the need to access and verify them requires a means to ensure that the architecture specification is kept consistent with the verification environment.

Each CSR is modeled in a functional environment referred to as the Register Access Layer (RAL). The RAL provides the ability to predict the value of any register at any time to act as a functional checker/scoreboard for an IP or the entire SoC. Further, it can dynamically determine the system address of any register even if the register is accessible over multiple access paths/interfaces. Finally, it offers a view to test writers to access the registers by way of consistent read/write API methods to model firmware operations while abstracting the lower-level hardware interfaces to enable reusable/portable test content. A validator can verify the functionality of every register and track register access and coverage through the RAL.

Each IP block contains one or more groups of register blocks/files within a local RAL environment. The RAL environment is associated with an IP (or sub-IP), cluster, or fullchip testbench environment. The hierarchical construction of Testbench environments allows the RAL environment at each level to be reused and each local RAL environment is created when its Testbench environment is created.Each RAL register file/block is accessible from the top-level RAL environment even though the actual construction of the register files/blocks is done at the sub-RAL environment level.

The process of creating/integrating the Register Access Layer (RAL) can be summarized as follows:

1. Define registers types and register addressing/organization in SystemRDL

2. Add inter-register behavior attributes, security groups, and special properties in SystemRDL

3. Use Intel specific tool to generate RAL collaterals each time the single-source RDL inputs are changed

4. Extend the tool-generated RAL environment to add constraints, overrides/customizations,

address generation functionality, and mapping of register frontdoor read/write access path(s) to specific RAL adapter/translation sequence(s)

5. Optional: Create a wrapper RDL on an IP/cluster/SoC level to define integration-level overrides of specific properties for registers and register groups

6. Associate the user-extended or wrapped RAL environment to a testbench top/sublevel environment

7. Define RAL adapter/translation sequence(s) for each frontdoor read/write access path

The CREST sequence is called from a wrapper OVM Test residing in the Saola/OVM environment. CREST first queries the RAL Environment to discover the set of registers to be tested. It then commands RAL to perform a sequence of reads and writes, which RAL implements by sending transactions to the simulated RTL. The responses received from the RTL are checked against the register models in the RAL Environment.

The diagram above shows CREST within OVM Test environment.

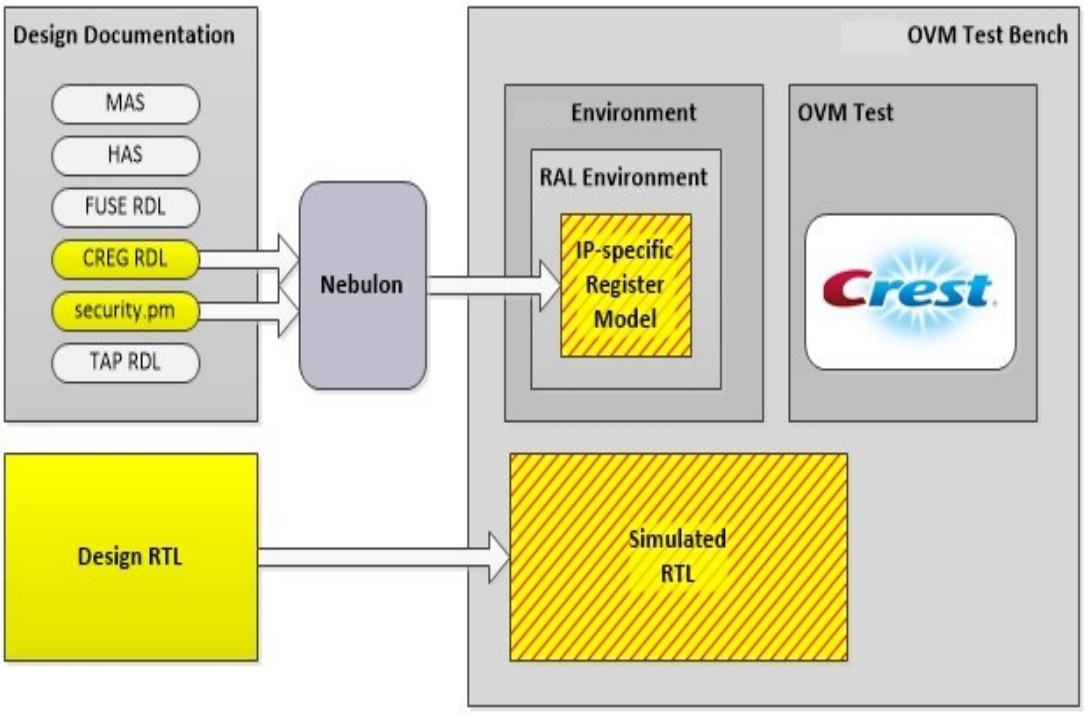


Figure 5.1: Crest flow

By thoroughly testing the registers documented in the RAL Environment, CREST builds confidence in the equivalence between the IP-specific Register Model in RAL and the simulated RTL. Because the IP-specific Register Model is auto-generated from the CREG RDL and security.pm file, CREST also raises confidence that the Design RTL matches the register spec (i.e., CREG RDL and security.pm).

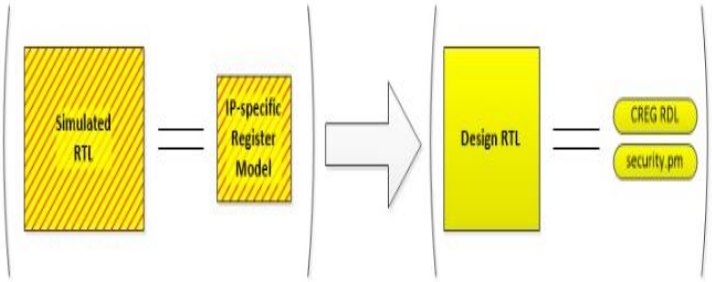


Figure 5.2: Crest

5.1 CREST Algorithms

5.1.1 Register Reset Testing

Algorithm:

- Reset IP
- For every register in the IP:
- Read the register using a SAI in the read access policy of the register
- Compare returned data against default value in RAL

Test covers reset value only, comparing RDL to RTL.It does not test register association with a particular reset domain. Register level black list enables skipping of registers that are known to be problematic.

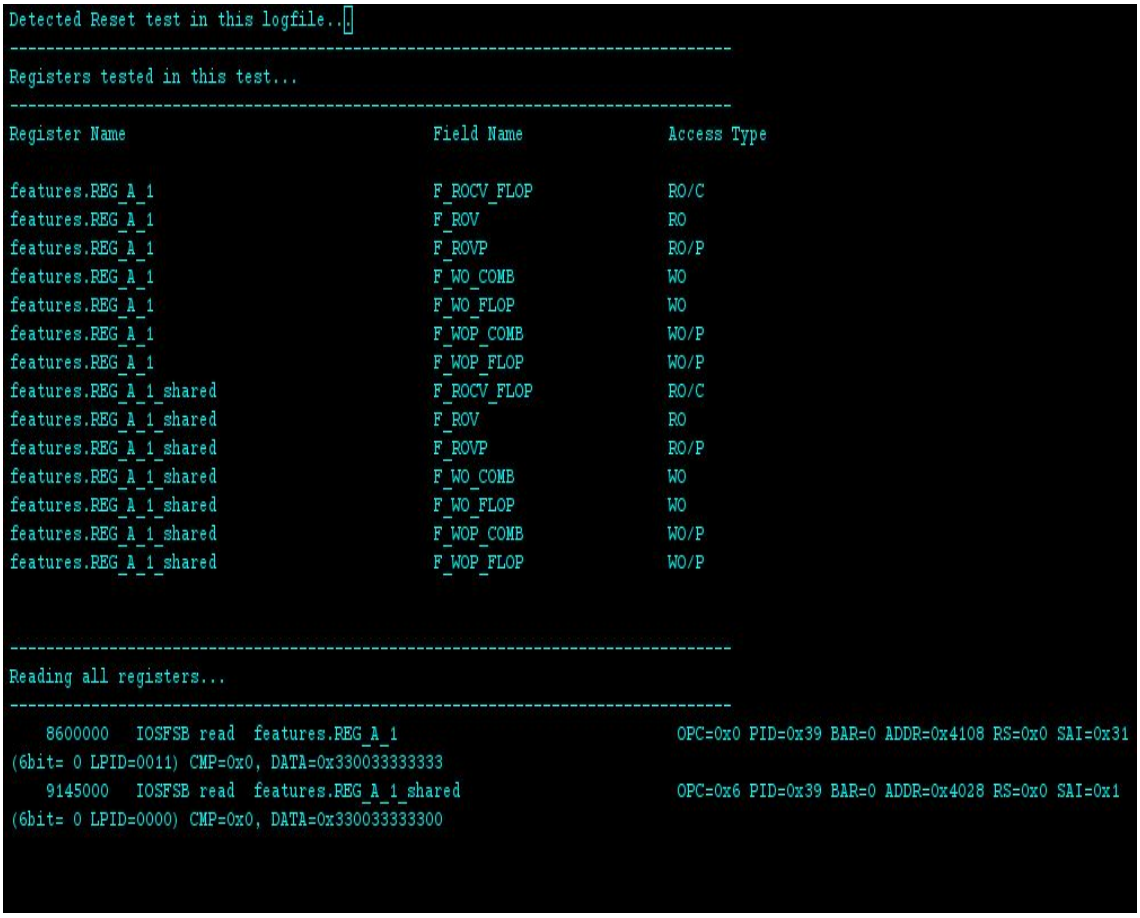


Figure 5.3: Register reset test

5.1.2 Field Attribute Testing

Algorithm:

- For every register in the IP:
- Select a random value to be written
- Write value using an SAI in the write access policy for the register
- Read back and compare against RAL prediction

Register level black list enables skipping of problematic registers Field level constraints specified via the VALLegalValues UDP are satisfied in the write value randomization. Because the write data is randomized, test coverage increases with repeated runs using different seeds.

```
--------------------------------------------------------------------------------
Registers tested in this test...
--------------------------------------------------------------------------------
--Register Name--                          --Field Name--          --Access Type--
features.REG_A_1                           F_ROCV_FLOP             RO/C
features.REG_A_1                           F_ROV                   RO
features.REG_A_1                           F_ROVP                  RO/P
features.REG_A_1                           F_WO_COMB               WO
features.REG_A_1                           F_WO_FLOP               WO
features.REG_A_1                           F_WOP_COMB              WO/P
features.REG_A_1                           F_WOP_FLOP              WO/P
features.REG_A_1_shared                    F_ROCV_FLOP             RO/C
features.REG_A_1_shared                    F_ROV                   RO
features.REG_A_1_shared                    F_ROVP                  RO/P
features.REG_A_1_shared                    F_WO_COMB               WO
features.REG_A_1_shared                    F_WO_FLOP               WO
features.REG_A_1_shared                    F_WOP_COMB              WO/P
features.REG_A_1_shared                    F_WOP_FLOP              WO/P
--------------------------------------------------------------------------------
```

Figure 5.4: Field attr test

```
Starting the test...
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
|Started Testing register features.REG_A_1
|   F_ROCV_FLOP[7:0]              RO/C
|   F_ROV[15:8]                   RO
|   F_ROVP[23:16]                 RO/P
|   F_WO_COMB[31:24]              WO
|   F_WO_FLOP[39:32]              WO
|   F_WOP_COMB[47:40]             WO/P
|   F_WOP_FLOP[55:48]             WO/P
--------------------------------------------------------------------------------
   8600000   IOSFSB read  features.REG_A_1                        OPC=0x0 PID=0x39 BAR=0 ADDR=0x4108 RS=0x0 SAI=0x31 (6bit= 0 LPID=0011)
CMP=0x0, DATA=0x330033333333
   9145000   IOSFSB write features.REG_A_1                        OPC=0x1 PID=0x39 BAR=0 ADDR=0x4108 RS=0x0 SAI=0x1  (6bit= 0 LPID=0000)
CMP=0x0, DATA=0x923a80dfbeb9b165
   9565000   IOSFSB read  features.REG_A_1                        OPC=0x0 PID=0x39 BAR=0 ADDR=0x4108 RS=0x0 SAI=0x2a (6bit=21        )
CMP=0x0, DATA=0x330033333300
   10145000  IOSFSB write features.REG_A_1                        OPC=0x1 PID=0x39 BAR=0 ADDR=0x4108 RS=0x0 SAI=0x30 (6bit=24        )
CMP=0x0, DATA=0xc2d87b0f3da55b9b
   10665000  IOSFSB read  features.REG_A_1                        OPC=0x0 PID=0x39 BAR=0 ADDR=0x4108 RS=0x0 SAI=0x30 (6bit=24        )
CMP=0x0, DATA=0x330033333300
```

Figure 5.5: Register A Attribute test

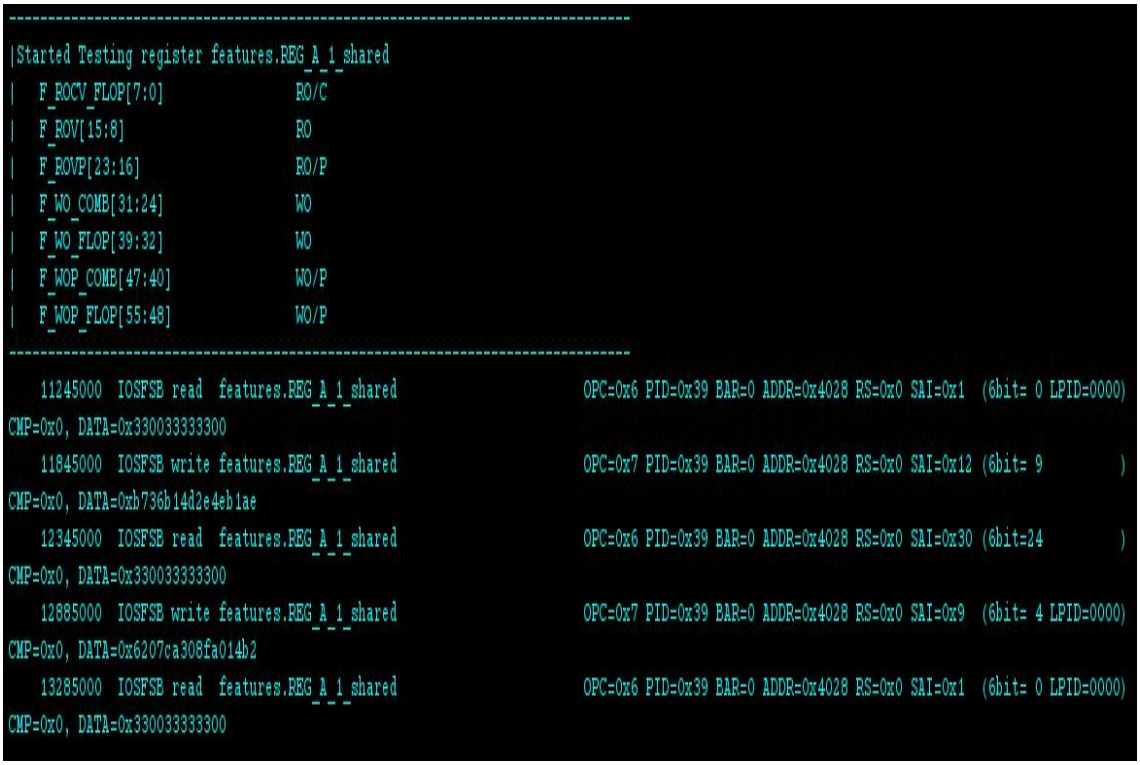


Figure 5.6: Register A_shared Attribute test

5.1.3 Lock Testing

Algorithm:

- Accumulate list of lock bits from all lockable fields
- For each lock bit:
- Attempt to toggle all associated lockable fields, read and compare against RAL prediction
- Attempt to toggle lock bit
- Attempt to toggle all associated lockable fields, read and compare against RAL prediction
- Attempt to toggle lock bit
- Attempt to toggle all associated lockable fields, read and compare against RAL prediction

Toggling lock bits twice is necessary to generically test both active low and active high locks.The reset domain of each lock bit is not tested RAL File, Register and Field level black lists allow skip-

ping fields that “kill” the IP (i.e., reset bits, PG control, etc.).Combination, multi-bit, and external locks are not supported; potential future enhancement.

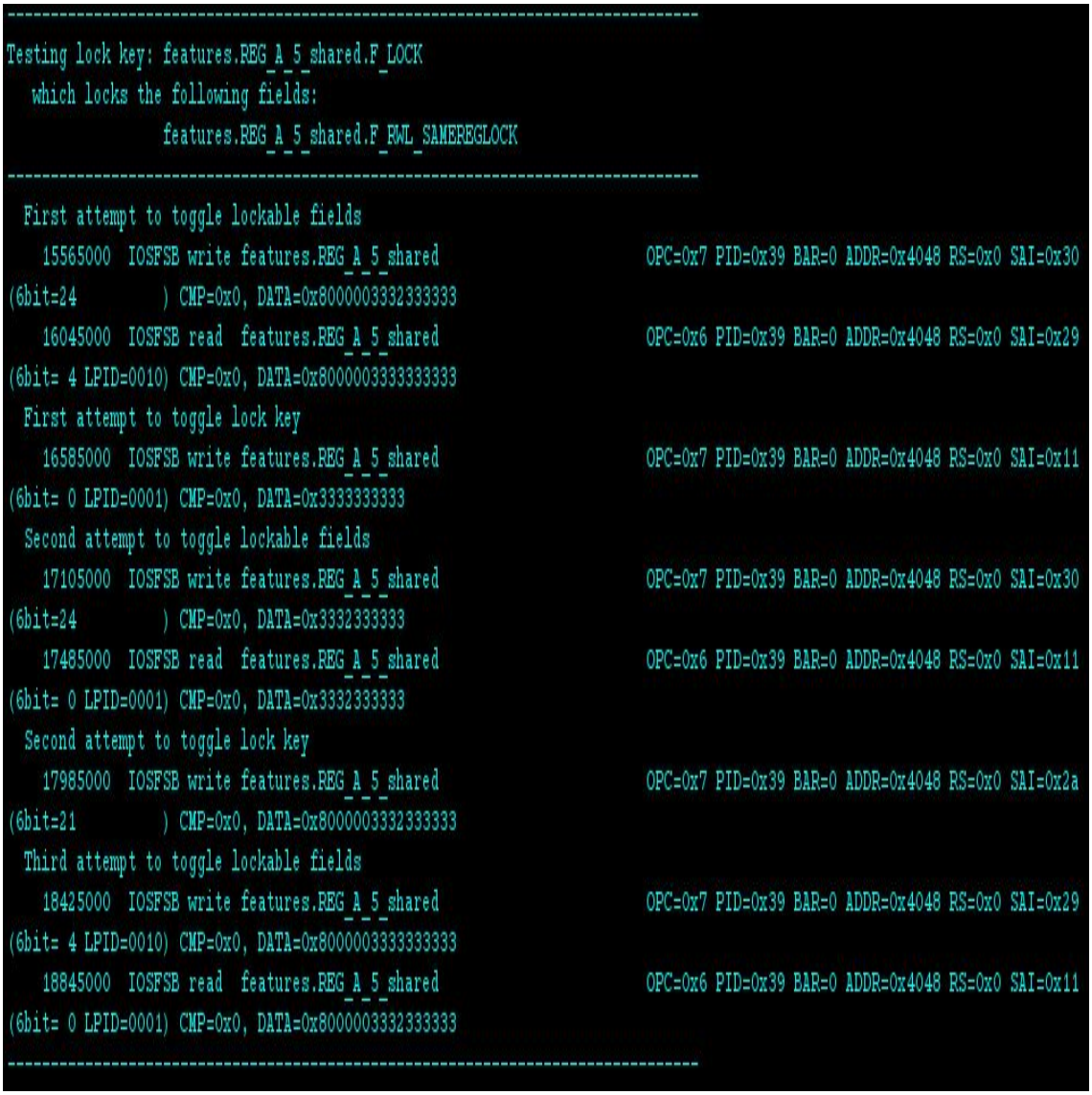


Figure 5.7: Lock test

Chapter 6

Conclusion and Future work

6.1 Conclusion

As IP integration is more prone to errors quality checks are mandatory to ensure efficient implementation ensuring IP/SoC protection. Testing of CR specification is done to ensure there are no mismatches between RDL and the RTL generated.

6.1.1 Future Work

- CREST:Shadow register testing
- CREST:SAI testing
- RDL/RAL/RTL mapping

Bibliography

[1] W. Kruijtzer et al., ”Industrial IP integration flows based on IP-XACT standards,” 2008 Design, Automation and Test in Europe, Munich, 2008.

[2] IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows,” in IEEE Std 1685-2009 , vol., no., pp.1-374, Feb. 18 2010.

[3] Julian Gorfajn, “RDL - Register Description Language”, Maxtor Corporation

[4] RDL (v1.0) Language Specification, ”Register description language” for the automation of documentation, verification, hardware, and software design 2002-2006,Cisco Systems , Inc.

[5] Victor Berman,” The P1685 IP-XACT IP metadata standard”, IEEE Council on Electronic Design and Automation,28 Aug.2008 pp.316-317.

[6] SystemRDL and Nebulon Tutorial Reference Guide, Intelpedia.

[7] www.accellera.org/downloads/standards/systemrdl.