# Comprehensive Register Description Languages

## The case for standardization of RDLs across design domains

David C Black
Doulos Inc
Austin, TX
David.Black@Doulos.com

Doug Smith
Doulos Inc
Austin, TX
Doug.Smith@Doulos.com

***Abstract*— Registers are pervasive in the design of electronics systems and components. Many systems contain literally hundreds of registers. The ability to quickly model registers in a variety of formats is extremely useful. There are many use cases for modeling registers ranging from software that access registers to the fundamental design and verification issues. Simply having a register accurate model of the hardware often suffices to allow early software design without many of the details behind the registers. It also happens that register descriptions and functionalities are remarkably simple to describe, and is often described in a tabular form in specifications. Mapping from register description tables to implementation is fairly straight forward, but tedious due to the sheer number of registers present in designs. Fortunately, since the number of variations is fairly small, the task is automatable. A formal Register Layer as promoted by UVM and others can help because it provides a basis to automated tools (i.e. EDA) to work from.**

**We open with a review of various use cases that benefit from a Register Description Language (RDL) and provide motivations for the development of a RDL. Use cases include System Level (architectural), Virtual System Platforms, verification, High Level Synthesis (HLS), RTL, verification, embedded software, validation and even design for test. This establishes a framework for discussion of features that need to be accounted for.**

**The paper then examines the features needed in order to support the use cases. For example, register and bit addressability, back-door access, notification, model performance and overhead. The goal is to broaden the view of registers from beyond any one discipline.**

**Currently, there are a number of propriety register solutions (some open source) available with a variety of input and output formats. For example, UVM provides a framework for creating registers as a part of the verification environment. Larger commercial vendors (e.g. Cadence, Mentor, Synopsys) and a variety of small commercial vendors (e.g. Duolog, Semifore, and others) supply frameworks, methodologies and tools to address the issues, but they tend to address niche solutions (i.e. a subset of the use cases) and use proprietary formats (i.e. not portable across domains). Additionally, the author is aware of numerous internal solutions by user companies. A few standards (e.g. Spirit IP-XACT, SystemRDL, and UVM), provide partial solution to this problem area as well. An overview of some current standards activities relating to RDLs will be presented (e.g. Accellera SystemRDL, OSCI's CCI). This paper provides an overview of these different solutions, relates them to the use cases, and considers their value.**

**We close with a look at attributes needed to make a more universal RDL standard, a successful approach for the EDA industry as a whole. For example, besides the ability to rapidly create models, and verification environments, RDL may benefit by providing a synthesizable framework when initiating new IP and verification frameworks to enable rapid validation of designs. There are also considerations for software and documentation. The paper concludes with suggestions for the standards community on where to take RDL with a goal of minimizing reinvention and maximizing reuse.**

***Keywords – Registers, ESL, Modeling, RTL, Verification, Documentation, CSR, RDL, UVM, RDL, IP-XACT***

## I. INTRODUCTION

Control and Status Registers (CSR's) or simply "registers" are a common feature in most electronic designs. This is particularly true of any design that includes processors and programming.

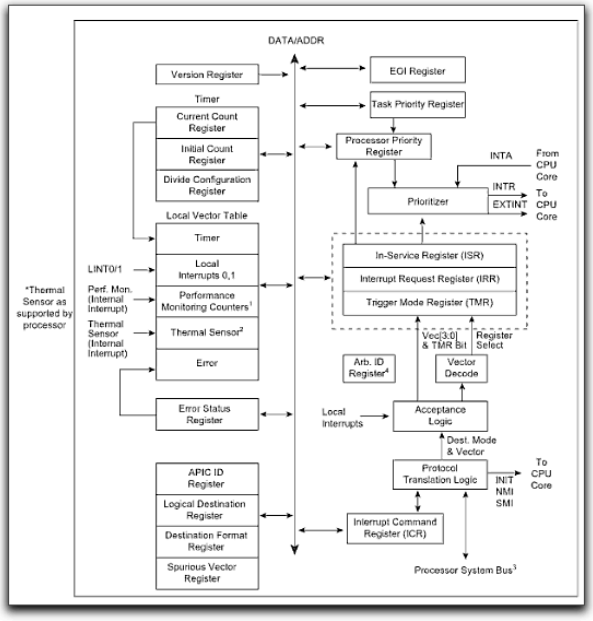


**Figure 1 Example of registers in a system**

Figure 1 is a common diagram of any modern system, and graphically spells out the presence of registers throughout evan a small portion of a design. It is not uncommon for designs to have hundreds of registers for the control and observation of hardware.

A CSR appears to a programmer as one or more memory accessible storage locations, but often with very different behavior than an ordinary memory. CSR's may be used as a means of data input or output. Writing to a CSR often controls the behavior of hardware devices in the system. Reading a CSR may obtain status information as the name implies. Oddly to some, even the act of reading may affect or control the operation of a piece of hardware (e.g. clearing status or initiating an operation). In fact, reading a CSR multiple times will often yield different information, leading programmers to consider a CSR as volatile unlike a normal memory.

Back in the 1980's, one of the authors was involved in a number of ASIC projects where it became painfully obvious that the specification of registers was a tedious task. In those days, we either did repetitive manual implementations or occasionally wrote simple scripts to help automate some of the process.

More recently, tools have begun to show up commercially to address these issues, but most of these tools show up in different application domains. For example, software tools help manage register management for embedded design, hardware designers have tools to help describe register description. Architects are starting to see tools that help with the specification. In most cases, the tools are somewhat narrow in their scope, and miss all the applications.

## II. TERMINOLOGY

Before proceeding much further it is good to establish some terminology.

CSR – Control and Status Register, a

ESL – Electronic System Level of abstraction includes TLM and above. Used by architects to describe a system at a more vague level than RTL. May include timing or may be untimed.

HDL – Hardware Design Language

HVL – Hardware Verification Language

RAL – Register Abstraction Layer

RDL – Register Description Language

Register – used by itself usually refers to a CSR

RTL – register transfer level of abstraction

SoC – System on a Chip

SVA – System Verilog Assertions

TLM – Transaction Level Modeling describes a model in terms of transactions and high level function calls rather than wires and low-level interconnect.

Verification – refers to the process of verifying that the hardware design (RTL) matches the specification. Focused on functional correctness and finding design bugs at a detailed level.

Validation – a process that occurs usually at the end of the design process using the real hardware to confirm performance and functional characteristics match the original specification.

## III. PROBLEM

Any discussion with other engineers will confirm that copying or cut-n-paste is a common engineering practice when dealing with any sort of repetitious design, and register design follows that pattern. This process is error prone.

Another concern is how pervasive registers are as they affect the design process in many application domains. Registers affect all of the following areas:

- Specification in multiple areas
- Hardware Implementation
- Software Implementation
- Verification
- Validation
- Documentation (sometimes even end-users)
- Power (retention and activity)
- Debug
- IP

There is a need for a solution that addresses all of the preceding in a consistent and compatible manner. It is important to recognize that these domains may use different "languages" including but not limited to:

Architects – C/C++, UML (possibly SysML), SystemC

Software – typically C/C++ or Java

Verification – SystemVerilog, PSL, e, Vera

Hardware – Verilog, VHDL, SystemVerilog

Of course it is also important to recognize natural language barriers in terms how architects, programmers and engineers understand the design issues. The goal is to assure no surprises, and these language issues can be fundamental causes.

Another problem has to do with the size and complexity of register designs. Modern designs often contain hundreds to thousands of registers in a single design. Subsystems may contain fewer, but even dealing with tens of registers is error prone. Part of the issue is simplifying the descriptions and making them more concise at the same time.

The acquisition of complex IP, whether internal or external, leads to even more problems because designers need to address issues with address maps and behaviors in designs not of their own making.

There are also problems with relationships among registers. Often there are dependencies on the order of access or interactions between configurations/status of different registers. These need to be expressed and understood by all the design domains. Along similar lines, addressing from some designs can be dynamic. In other words, the register address may be configurable via another register, or possible based on external switching or connections.

Finally, there is the issue of design maintenance. As a design progresses, there are often design changes to registers. Additions or modifications to the registers need to be propagated to the entire design hierarchy consistently.

To summarize the problems we see:

1. Error prone process
2. Inability to share across design domains
    a. Design language
    b. Domain language
3. Sheer size dictates need to simplify description
4. Problems with integrating and collaborating with register designs from different sources
5. Problems communicating register interactions
6. Problems with incremental updates to the entirety

## IV. History & Literature

It is worth looking as some of the available history and literature on this subject.

The earliest paper I could easily locate was a paper presented to the Synopsys User's Group (SNUG) conference in 2006 by Julian Gorfajn, an employee of Maxtor at the time. The paper outlined two generations of tools used internally to manage the problems described here. They created two different input languages and then generated a variety of outputs. The tools and specification remained proprietary.

A year later, a paper presented by Cisco at DVCon outlined another internal tool effort. It was strikingly similar to the Maxtor effort; however, apparently quite independent as confirmed by telephone interviews conducted with the authors of both papers. One very positive aspect of the Cisco effort is that Cisco recognized it was not in their best interest to develop EDA tools and they wanted to divest themselves of maintenance and perhaps create an eco-system to support the concept. This resulted in two transfers of the technology. First, they transferred the tool to Denali, but with the stipulation that the input format be standardized. As a result, they started the SystemRDL effort with Accellera.

In 2005, the standard was formalized and became an official Accellera standard; however, it has yet to be submitted to the IEEE. In addition to the formal standard, there is an ANTLR specification available for download. ANTLR is a language used to specify languages that provides a modern alternative to the tried and true LEX and YACC.

The book Hardware/Firmware Interface Design ©2010 by Gary Stringham provides some excellent advice on how hardware and software teams should work together, and lots of best practices for hardware. It should probably be required reading for anybody involved in the specification and design of hardware.

## V. Available Solutions

There are a number of available tools and technologies that offer partial solutions, but all of them tend to fall somewhat short of a comprehensive solution. This is partly because they each embrace a proprietary input format, which tends to exclude the ability to share designs with groups that do not use the selected technology. This section explores some of the issues with each of these offerings. The URL www.garystringham.com/rdt.shtml provides an excellent collection of this information focused on tools. Here is a summary with a focus on the input languages used to describe registers and the outputs available.

### *A. Open Source*

Reggen – An Google project that creates RTL from an XML format taken from stylized Excel spreadsheets. Not mature.

UVM RGM – A Cadence package released under Apache 2.0 license that contains an implementation of the UVM Register Layer and a converter that reads IP-XACT and outputs SystemVerilog utilizing the UVM Register Layer. There is an implication where System RDL is supported.

UVM Register Layer – The register layer of UVM is not a register description language, but rather an API to support verification of registers in the universal verification methodology.

### *B. Standards*

IEEE 1685 IP-XACT – A standard created by the SPIRIT consortium (now Accellera) that defines a schema for describing components with their registers for configuration and integration. The IP-XACT format has a limited syntax for describing registers, but it can be extended using proprietary vendor extensions.

SystemRDL – A standard created by Accellera, which includes an ANTLR description. No tools are provided; however, the ANTLR provides a good basis for the creation of tools.

SystemC CCI – Currently under development in the Accellera Systems Initiative, the Configuration, Control and Inspection standard represents a standard API for tools and models to configure and control parameters in simulation models. While not directly addressing registers, there have been uses of this to set parameters that are often put into device registers.

### *C. Commercial*

Duolog Bitwise – A graphical, Java-based tool built upon the Eclipse platform for graphically entering and maintaining register descriptions in a design. Bitwise read in SPIRIT IP-XACT format, and outputs most common output formats, including generating documentation for functional specifications.

Semifore CSRcompiler - A command-line tool that reads in register descriptions in IP-XACT, SystemRDL, Excel, or CSRSpec language. Like other commercial tools, various target outputs are supported and documentation and verification models can be generated.

Cadence Verification Builder – A graphical tool for building verification components including register models. Supports multiple languages for outputs like *e*, SystemVerilog, and SystemC.

Mentor Certes Testbench Studio (Register Assistant) – A graphical tool that can read in CSV, IP-XACT, or XML and generate a UVM register model and documentation.

Agnisys IDesignSpec – A register model generator supporting input formats like SystemRDL, IP-XACT, XML, and CSV. Many outputs can be generated and IDesignSpec can be used as a command-line tool or office-suite plug-in.

Synopsys Ralgen – A command-line tool that reads in a register description in IP-XACT or RALF format. Provided free for VCS simulator users or with Synopsys' VMM open source download.

## VI. Desired Characteristics

Several desired characteristics are needed in any real solution:

1. Flow and ease of integration
2. Simplicity and learning curve
3. Cost of acquisition
4. Cost of maintenance
5. Flexibility
6. Outputs
    a. Documentation (RTF, XML, MIF)
    b. Synthesizable RTL (Verilog)
    c. SystemC TLM for Virtual Platform
    d. Verification (SystemVerilog, UVM, OVM)
    e. Software definitions (C/C++, Java)
7. Standards compliance

The flow we are trying to achieve is illustrated in the following diagram. Although, many tools allow for multiple input sources, we believe a single source is the ideal situation. By using a single standard, it is easier when exchanging IP to expect a single format from the provider, and it is easier to understand the system when there is a single specification to learn.

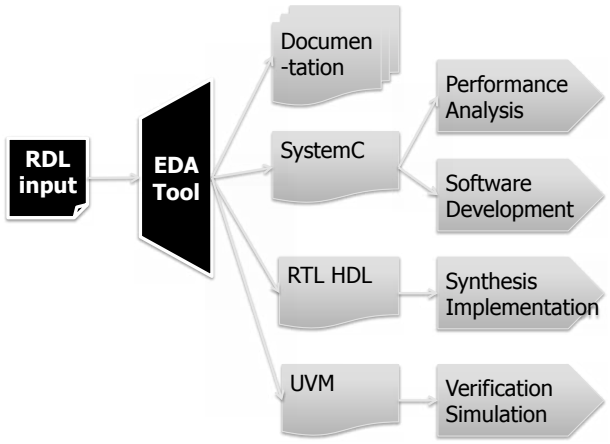


**Figure 2 Tool Flow based on RDL**

Flow and ease of integration refers to the manner in which a solution fits into existing design workflows and processes. Many proprietary/private solutions start with the documentation and move towards implementation. This is somewhat problematic in that they usually presume a word processor as the original input, and a formatting style to go with it. The common word processor selections include Microsoft Word, Adobe Framemaker, Open Office, or an XML editor. By contrast, a formal register description language (RDL), would allow for plain text from any source. XML is somewhat clumsy for a human to enter by hand and presumes some sort of entry tool to check for consistency. A more ideal solution to this characteristic is probably a more formalized syntax such as an RDL.

An RDL description must be able to:

1. Provide register groupings (register blocks)
2. Provide a register name
3. Identify bit fields
4. Specify reset characteristics
5. Specify read/write characteristics of fields for software including volatility, persistence, illegal, and ignored
6. Specify read/write characteristics with respect to hardware
7. Identify the addressing from various points of view (even from masters on different buses with bridges)
8. Allow description of interactions between registers (e.g. access ordering requirements and exclusivity)
9. Specify streaming (e.g. two registers that are written to in a ping-pong fashion)
10. Specify time ordering aspects (one register must be read before the second)
11. Identify component instances in the hardware
12. Allow for reuse of a description of a block definition (same register block may have several copies)

Furthermore, an RDL must be text only to allow simple entry, have no dependence or restrictions from a GUI, be usable for IP exchange, and encourage a tool support ecosystem.

Consider the following two descriptions. The first is an IP-XACT description, and the second is SystemRDL. Notice how much simpler the SystemRDL is to read when presented as text.

```
<?xml version="1.0" encoding="UTF-8" ?>
<spirit:memoryMaps
xmlns:spirit="http://www.spiritconsortium.org/XMLSchema/SPIRIT/1.4"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.spiritconsortium.org/XMLSchema/SPIRIT/1.4
http://www.spiritconsortium.org/XMLSchema/SPIRIT/1.4/index.xsd">
 <spirit:memoryMap>
  <spirit:name>some_register_map</spirit:name>
  <spirit:displayName>RDL Example
  Register</spirit:displayName>
  <spirit:addressBlock>
  <spirit:name>some_register_map</spirit:name>
  <spirit:displayName>RDL Example
  Register</spirit:displayName>
  <spirit:description>This address map
  an example register.</spirit:description>
   <spirit:baseAddress>0x0</spirit:baseAddress>
   <spirit:range>0x2000</spirit:range>
   <spirit:width>32</spirit:width>
   <spirit:usage>register</spirit:usage>
   <spirit:volatile>true</spirit:volatile>
   <spirit:register>
    <spirit:name>chip_id_reg</spirit:name>
    <spirit:displayName>This chip part
    number and revision
    #</spirit:displayName>
    <spirit:description>This register
    cotains the part # and revision #
    for XYZ ASIC</spirit:description>
    <spirit:addressOffset>0x0</spirit:addressOffset>
    <spirit:size>32</spirit:size>
    <spirit:volatile>true</spirit:volatile>
    <spirit:access>read-write</spirit:access>
    <spirit:reset>
     <spirit:value>0x12345671</spirit:value>
     <spirit:mask>0xffffffff</spirit:mask>
    </spirit:reset>
    <spirit:field>
     <spirit:name>rev_num</spirit:name>
     <spirit:description>This field
     represents the chips revision
     number</spirit:description>
     <spirit:bitOffset>0</spirit:bitOffset>
     <spirit:bitWidth>4</spirit:bitWidth>
     <spirit:access>read-only</spirit:access>
    </spirit:field>
    <spirit:field>
     <spirit:name>part_num</spirit:name>
     <spirit:description>This field
     represents the chips part
     number</spirit:description>
     <spirit:bitOffset>4</spirit:bitOffset>
     <spirit:bitWidth>28</spirit:bitWidth>
     <spirit:access>read-only</spirit:access>
    </spirit:field>
   </spirit:register>
  </spirit:addressBlock>
 </spirit:memoryMap>
</spirit:memoryMaps>
```

Figure 3 Example Register definition using IP-XACT

```
addrmap some_register_map {
  name = "RDL Example Register";
  desc = "This address map contains one example register";
  reg chip_id {
     name = "This chip part number and revision #";
     desc = "Contains the part # & revision # for XYZ ASIC";
     field {
       hw   = w;
       sw   = r;
       desc = "Field represents the chips part number";
     } part_num[31:4] = 28'h12_34_56_7;
     field {
       hw   = na;
       sw   = r;
       desc = "Field represents the chips revision number";
     } rev_num[3:0] = 4'b00_01;
  };
  external chip_id chip_id_reg @0x0000;
```

Figure 4 Example Register definition using SystemRDL

It may be argued that XML editing tools can make the IP-XACT look simple as well; however, it is well known practice within the industry for engineers to edit the XML directly. Because of the complexities of XML, it is easy for engineers to make a mistake. Also, all XML editing tools are not alike; whereas, text formats and their editors are well known throughout the engineering industry.

## VII. Conclusion

Several useful register implementations are available today, but implementations are not idyllic because they tend to target too specific of application domains and they fail to embody an industry standard. Of the available industry standards, SystemRDL comes the closest to meeting the desired characteristics and we recommend it be considered as the RDL language of choice for the numerous commercial generators and open-source implementations. However, SystemRDL may have some complexities that are difficult, impossible to use, or even overkill for meeting the ideal requirements. While we are not advocating a new standard since SystemRDL is a comprehensive solution, we suggest that a variant or subset of SystemRDL may be order—one that distills SystemRDL down to just the salient set of features required—and that Accellera make a more concerted effort to promote and standardize that RDL across its various subcommittees.

## VIII. References

[1] Faust, J. Michael. "The Register Description Language as a Foundation for Modern System Design," Cisco Systems Whitepaper, 2006.

[2] Gorfajn, Julian. "RDL – Register Description Language," Maxtor Corporation, 2005.

[3] "IEEE 1685-2009: IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows," IEEE Computer Society, 2009.

[4] Stringham, Gary. Hardware/Firmware Interface Design: Best Practices for Improving Embedded Systems Development. Elsevier, 2010.

[5] "SystemRDL v1.0: A specification for a Register Description Language," The SPIRIT Consortium, 2009.

[6] Barry, Peter and Crowley, Patrick "Modern Embedded Computing: Designing Connected, Pervasive, Media-Rich Systems" Elsevier , 2012