Using Automation to Close the Loop Between Functional Requirements and Their Verification

Brian Craw
Cypress Semiconductor
Brian.Craw@cypress.com

David Crutchfield
Cypress Semiconductor
David.Crutchfield@cypress.com

Martin Oberkoenig
Cypress Semiconductor
Martin.Oberkoenig@cypress.com

Markus Heigl
Cypress Semiconductor
Markus.Heigl@cypress.com

Martin O'Keeffe
Cypress Semiconductor
Martin.O'Keeffe@cypress.com

***Abstract*- Managing and verifying the requirements that make up modern complex digital designs is becoming an increasingly burdensome task. This combined with the need to meet industry safety standards such as ISO 26262 means the old-school method of manually tracking closure in spreadsheets is no longer feasible. This paper describes a flow centered around a requirements management tool that will aid in the gathering of requirements, generate System Verilog coverage code used to prove those requirements, and back-annotate simulation results into the tool to close the loop.**

I. INTRODUCTION

Tracking the fulfillment of requirements is an integral part of the verification process designed to ensure the requirements specified by the stakeholders such as Marketing, Architecture, Design, and Verification groups are fulfilled by the end design. At Cypress, the Verification Engineer is responsible for understanding the requirements, extracting them from the source documents, reviewing them with the designers, writing tests that exercise the desired functionality, and finally linking the System Verilog coverage results back to the requirements to prove their fulfillment. All requirements information is stored in a spreadsheet and a manual process is used to ensure each requirement is met by System Verilog coverage items and/or directed tests. Any changes to the requirement, coverage code, or simulation results needs to be noticed and updated manually. This manual process is time consuming and error prone.

Also, a key part of meeting industry functional safety standards such as ISO 26262 is maintaining traceability from high-level customer requirements down to the verification requirements proving the customer requirements have been met.

For these reasons, we realized the need to put in place an enterprise quality requirements management tool that was not only capable of managing our high-level marketing requirements all the way down to verification requirements but also extensible enough to enable automation in the area it would be most beneficial: functional verification.

II. DEFINITION OF TERMS

Coverage Intent	A high-level description of System Verilog coverage code.
CovGen	System Verilog Coverage Code Generator – Internally developed tool for reading in a spreadsheet that defines coverage intent and producing System Verilog covergroups, coverpoints, and crosses.
ISO 26262	International functional safety standard for electronic systems within an automobile.
RM Tool	Siemens Polarion - Requirements Management Tool
SV	System Verilog
UCDB	Unified Coverage Database

VMS	Verification Management System – Internally developed tool for compiling, optimizing, running simulations and collecting the results.
Questa VRM	Questa Verification Run Management – Mentor's tool for launching verification tasks within a regression.
Work Item	Describes an item stored in the RM Tool

III. BACKGROUND

A. ISO 26262

Functional safety as defined in the ISO 26262 for road vehicles is a must-have for most automotive systems. The high integration and automation of functionality provided by electronic control units is increasing tremendously. For those system's high-level requirements, so-called safety goals, are derived through a hazard analysis and risk assessment. The safety goals are refined through functional and technical requirements into hardware requirements for the actual implementation. To ensure that the design meets the original safety goals traceability is very important.

A Requirements Management tool enables us to cope with the complexity of such systems, ensuring that requirements are not forgotten at a certain abstraction level and all intended functionality is finally present in the design. More than that, the design must also work as intended, which means all requirements for a certain design must be verified.

The less manual steps exist in such a flow the less systematic failures can happen during development.

B. Requirements Gathering and Consolidation

A Requirements Management tool manages a database that stores customizable items of information called work-items, maintains the relationships between those items, is capable of custom workflows around those items, and maintains a history of changes to those items and their relationships to one another.

With these features, we can define a flow for managing our requirements from the highest levels, the Customer Requirement, down to the lowest level, the Verification Requirement, and greatly reduce the effort involved in managing the thousands of requirements that can make up a complex piece of IP.

Reality shows that the whole requirements elicitation process only works with a well-defined flow and tool support. In modern, worldwide operating companies it is already a challenge to collect requirements from different customers and put them into a central database. The requirements from different customers might be similar but can also be contradicting. So, a consolidation step is required to resolve conflicts and merge items. A Requirements Management tool can support handling a huge number of items, in enforcing a certain workflow for such tasks, and in reporting the status of such activity.

C. Tool Requirements

Even though the initial use of the Requirements Management tool is to track functional verification requirements the tool must support a wider breadth of features such that, in the future, its use can be expanded to other areas of the company such as mixed-signal IP development, back-end characterization and testing, firmware and software development, etc.

Keeping these long-term objectives in mind the following high-level requirements were decided upon.

1.) Usable across business units by the various requirements stakeholders

The tool must be usable by multiple types of users in the company. This means people from Marketing, Architecture, Design, and Verification should all have access to the tool and have limited knowledge of functional verification or functional verification tools.

2.) Usable across business sites/geographies

The tool must be usable across geographies without the need for continuous sync'ing of databases between sites. A Cloud-based solution is preferable due to the ease of access across sites as well as the added feature of not having to maintain hardware to contain the database and serve the tool.

3.) *Configurable, Customizable, and Extensible*

Configurable means the tool can be configured out-of-the-box via a GUI or command line to meet the majority of the requirements placed on the tool. Some configurable features would be setting up custom fields in a requirement work-item, creating a custom work-flow for different work-item types, creating reports showing the current state of all work-items in a certain project.

Customizable means the tool can be modified to add functionality that doesn't exist out-of-the-box. In our case, we wanted the ability for the tool to import a coverage results report, find the work-items corresponding to the coverage constructs in the report, and update their coverage state accordingly. This is not a feature supported out-of-the-box but is something that can be implemented using the tool's API.

Finally, an extensible tool is one that can be "hooked" to other tools to extend its own function. For example, it is natural to tie requirements management into other systems such as a bug-tracking tool. This allows a bug filed against a function of a piece of IP to be reflected in that feature's requirement work-item(s) thus possibly changing its state from, for example, "Verified" to "Needs-Review".

Managing functional requirements for logic designs is a large task and there are many EDA Vendor solutions targeted directly to that area. These tools provide integration directly into a Vendor's digital simulation environment. However, these tools strengths are not as a centrally accessible repository used for collaboration amongst various groups within the company. To provide a company wide solution we decided on a Requirements Management tool that provided the out-of-the-box customization and extensibility which will not only enable us to tie it directly into our verification flow via the coverage intent export and the coverage result import extensions but also enable future customization that will be of benefit to other areas of the company. The benefits of having a "single source of truth" for our requirements out-weighed the benefits of ease-of-integration into our verification environment. Ultimately, the tool chosen that met these requirements was Polarion from Siemens PLM.

D. Verification Management System (VMS) [1]

Years ago, Cypress developed VMS (Verification Management System) to manage logic verification regressions across the entire company. VMS leverages Mentor Graphics' Questa VRM (Verification Run Management) tool to launch compilation and simulation jobs through a load sharing facility (LSF), and to generate and merge functional coverage information. This system produces a single merged coverage database file (UCDB) containing the coverage information from all the simulations in a regression. Coverage information is extracted from this UCDB file, placed in an XML formatted file, and provided to the Requirements Management tool for back-annotation as will be described later in Section III Sub-Section F.

E. CovGen, a System Verilog Coverage Code Generation Tool

At the end of the requirements definition process a Verification Engineer has the task of writing functional coverage code to meet each of those requirements. The process of writing these coverage items can be lengthy and involve hundreds if not thousands of lines of System Verilog code. Constant iteration over this code to make updates and fix syntax errors can consume a significant portion of the verification cycle. To reduce this effort, we have integrated a home-grown Coverage Code Generator into our flow. The Coverage Generator takes as input a spreadsheet that defines the "intent" of the desired System Verilog coverage code. Each row in the spreadsheet describes a coverpoint or cross. The columns correspond to various fields that describe the specifics of the coverage construct. This intent is parsed and syntactically correct System Verilog coverage code is produced consisting of covergroups, coverpoints, and crosses.

This paper will describe how the coverage intent spreadsheet format is mapped to custom work-items within the RM Tool and how those items are used to close the loop between an item in the RM Tool and actual regression results.

III. FLOW

The Verification Requirement Tracking Flow is shown in Fig 1. Details describing each step are documented in the following sub-sections.

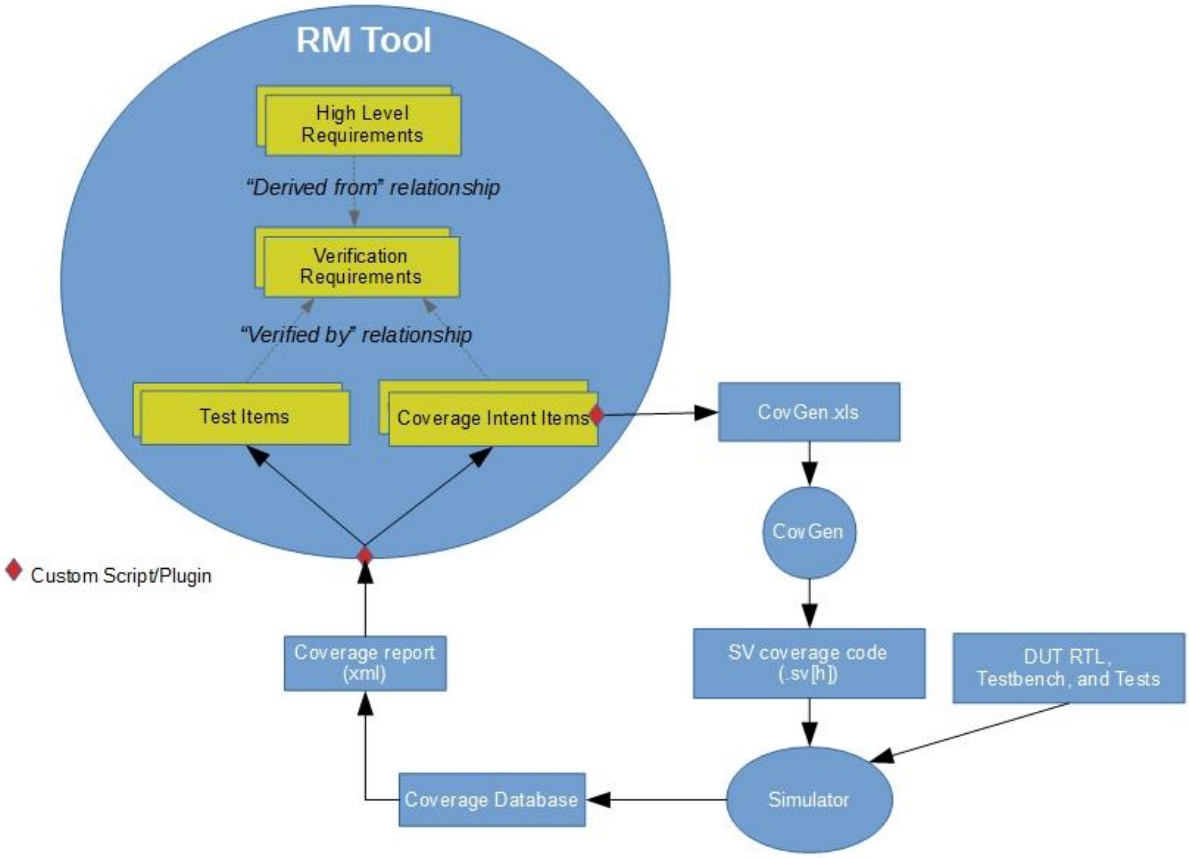

Figure 1. Requirements Tracking Flow

A. High Level Requirements

High-level requirements come from various sources such as customers, industry standards, internal architecture and design groups, etc. These high-level requirements may potentially have a one-to-many relationship to the Verification Requirements that verify them.

B. Verification Requirements

Verification Requirements are the final derivation of a high-level requirement. They should be atomic and measurable which makes it easy for them to directly correspond to an SV coverage construct such as a Coverpoint or Cross.

A single higher-level requirement may produce many lower-level Verification Requirements. For a higher-level requirement to be considered "covered" all derived Verification Requirements must be in their "covered" state: Coverage Intent items show 100% bin hits and Test items show a PASS.

All Verification Requirements must link back to a higher-level requirement. Any requirements that do not link to a higher-level requirement should be reviewed to determine if they are unnecessary or if a requirement is missing.

Table 1 shows an example of some of the fields that make up a Verification Requirement work-item in the RM Tool.

Field	Description
Title	The title of the requirement.
Source	The initial source of the requirement
Description	A description of the work-item.
Comments	Comments between users during the life of the work-item.
Category	The category the requirement falls under.
Assignee	The owner of the work-item.
Author	The author of the work-item.
Status	The current state of the work-item.
Linked Work-items	The work-items linked to this work-item, their relationship to it, and their states. i.e. the Coverage Intent work-items and if they are in their "covered" state or not.
Approvals	Who has approved the work-item.

Table 1: Verification Requirement Work-item Fields

C. Coverage Items

The coverage intent information is stored within the RM tool as "Coverage Intent" items whose fields correspond to columns describing the coverage intent in a spreadsheet. These items are exported as an XLS document and used by the CovGen utility to generate System Verilog code.

Table 2 shows an example of some of the fields that make up a Coverage Intent work-item in the RM Tool. Note that the Name field is auto-generated from other fields within the work-item (namely the Register and Field fields). This ensures a standard naming convention that the Coverage Generator and the coverage results import script can rely upon.

Field	Description
Title	The title of the requirement.
Description	A more detailed description of the work-item.
Assignee	The user assigned to the work-item.
Author	The author of the work-item
Status	The current state of the work-item.
Type	The type of coverage intent item: COVERPOINT or CROSS.
Category	The category of the coverpoint.
Group	Block in which the register is located.
Register	The register name.
Field	Register field or variable name.
Bits	Number or range of bits in field.
Qty	The number of instances of the field/variable.
Name	Name of coverpoint or cross. This field is also used when importing coverage results to find the correct work-item in which to apply the results.
Wgt	The weight of the coverpoint or cross.
Bins	The specification of what bins to create for the coverpoint.
CP1-CPn	If Type is Cross, then these fields contain the names of existing coverpoints used to create it.
Coverage	The back-annotated coverage number for this item. i.e. usually a percentage bin hits.

Field	Description
Comments	Comments between users during the life of the work-item.
Linked Work-items	The work-items linked to this work-item such as Test and Coverage-Intent items.
Approvals	Who has approved the work-item

Table 2: Coverage Intent Work-item Fields

Two other custom work-items have also been defined: The Test work-item and the Assertion work-item. The Test work-item corresponds to directed tests in the IP's regression suite. There are cases where a directed test is used to prove a condition that is difficult to reach with constrained randomization. In these cases, a requirement can be proven with a passing directed test. The Test work-item uses a standard work-item layout with the only custom field being the Result field which simply indicates if the test passed or failed. The Assertion work-item can be used to track assertion firings for functional coverage. The Coverage Generator only generates Coverpoints, Crosses, and Covergroups. It does not produce Assertions. A future enhancement may be to add this capability. Until that point, Assertions are tracked in a standard work-item type that simply logs whether the assertion has "fired". The results of a regression can still be back-annotated to the corresponding work-items if the "Name" field in the work-items matches the "Name" of the assertion in the SV code.

With these new work-items (Coverage Intent, Test, and Assertion) we now have the means to directly connect functional coverage results and test status to the requirements they meet. See Figure 2 for an example diagram showing the relationships between these work-item types.

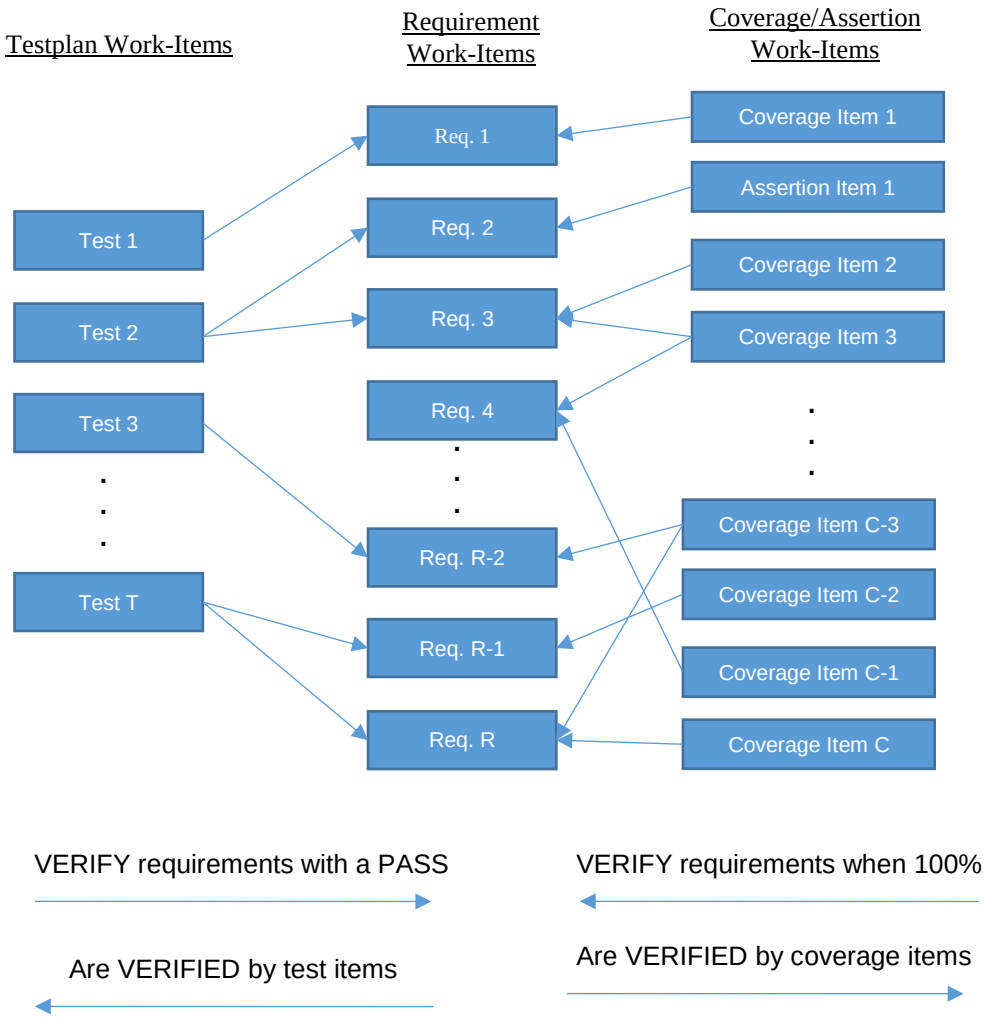

Figure 2: Relationships between Verification Requirements, Test Work-items, and Coverage Work-items

D. Export Coverage Intent

The customizable nature of the RM Tool was taken advantage of via a custom extension which allowed the Coverage Intent work-items to be exported into a spreadsheet (XLS) in the format used by the Coverage Generator.

The coverage intent fields in “Table 2: Coverage Intent Work-item Fields” will each correspond to a column in the coverage intent spreadsheet. “Table 3: Sample Coverage Intent Spreadsheet Format” shows some example rows from a Coverage Intent spreadsheet that defines a Covergroup, two Coverpoints, and a Cross.

CG_SAMPLE											
COMMENT				**WGT**							
This is a CovGen example				2							
Type	**Category**	**Group**	**Register**	**Field**	**Bits**	**Qty**	**Name**	**Wgt**	**Bins**	**CP1**	**CP2**
CP	Reg	SEQ	CTL	EN	23:20	2	cp_ctl_en	1	MIN, MAX, BIN='h8		
CP	Var			MODE	3:0	1	cp_mode	2	WALKONES		
CS							cs_cp_ctl_en_cp_mode	1	IGNORE{ CP1=MIN & CP2=15}	cp_ctl_en	cp_mode

Table 3: Sample Coverage Intent Spreadsheet Format

The Coverage Generator Tool processes the spreadsheet along with a configuration file and System Verilog coverage code is produced.

The first row in Table 3 informs the generator that the following rows describe a Covergroup and provides the Covergroup name after the 'CG_' prefix. The next row provides any desired type_options for the Covergroup. The last four rows contain the coverage element header and describe the Coverpoints and Cross that will produce the following System Verilog code. Note that only a subset of the Coverage Generator's capabilities is described here.

```
covergroup SAMPLE_cg;
  type_option.weight = 2;
  type_option.comment = "This is a CovGen example";

   cp_ctl_en_0: coverpoint bit_4_t'(ctl[0].peek_field_by_name("EN")) iff (sample_cp_ctl_en_0) {
    type_option.weight = 1;
    bins bin_min = {0};
    bins bin_max = {15};
    bins bin_8 = {8};
  }
  cp_ctl_en_1: coverpoint bit_4_t'(ctl[1].peek_field_by_name("EN")) iff (sample_cp_ctl_en_1) {
    type_option.weight = 1;
    bins bin_min = {0};
    bins bin_max = {15};
    bins bin_8 = {8};
  }
  cp_mode: coverpoint MODE iff (sample_cp_mode) {
    type_option.weight = 2;
    bins bin_walkone_0 = {4'b0001};
    bins bin_walkone_1 = {4'b0010};
    bins bin_walkone_2 = {4'b0100};
    bins bin_walkone_3 = {4'b1000};
  }
  cs_cp_ctl_en_0_cp_mode: cross cp_ctl_en_0, cp_mode  {
    type_option.weight = 1;
    ignore_bins bin_cp1_0_cp2_15 = (binsof(cp_ctl_en_0) intersect {0} && binsof(cp_mode) intersect {15});
  }
  cs_cp_ctl_en_1_cp_mode: cross cp_ctl_en_1, cp_mode  {
    type_option.weight = 1;
    ignore_bins bin_cp1_0_cp2_15 = (binsof(cp_ctl_en_1) intersect {0} && binsof(cp_mode) intersect {15});
  }

endgroup : SAMPLE_cg;
```

Figure 3: Example System Verilog code produced by Coverage Generator

E. Run regressions and gather coverage results

The generated SV code is added to the testbench and regressions are run. The Cypress Verification Management System (VMS) is used to run the regressions and merge all the collected functional coverage information into a single coverage database file.

F. Generate report from coverage database

The Simulator has a utility that can generate an XML report from the coverage database. This report contains all the coverage information related to the coverage constructs (Crosses, Coverpoints, and Assertions) used within the test environment as well as test names and pass/fail status.

G. Import coverage report into RM Tool and back-annotate Coverage/Test items

Another custom extension to the RM Tool written using the provided API is used to import the generated XML coverage report. The script parses the XML and finds all coverage constructs as well as test names and their status. The Coverage Intent and Test items names within the RM Tool are searched and matched with their corresponding items from the coverage report. Fields within the items are updated to show the status from the coverage report. Coverage Intent items are updated with their bin hit percentages and Test items are updated with their PASS/FAIL status.

A Verification Requirement is deemed as met when all Coverage items linked to it show 100% bin hits and all Test/Assertion items linked to it are marked as PASS.

H. Generate reports in the RM Tool to show state of Verification Requirements coverage

Another key feature of the RM Tool is the ability to generate customized reports. Each project in the RM Tool contains a dashboard showing some of these reports. These reports show the current state of the Verification Requirements coverage.

IV. CONCLUSION AND FUTURE WORK

The purpose of this paper is to document the approach to requirements management taken by Cypress with an emphasis on functional verification. This flow and a central "single source of truth" database of requirements within the tool enables the traceability and quality demanded not only by industry standards such as ISO 26262 but also the internal quality metrics our company enforces upon itself.

It is important to note that the tool and flow implemented around it discussed in this paper enable the requirements stakeholders within the company to interact with requirements in an efficient way focused on collaboration and traceability without being tied to a specific functional verification EDA vendor. Making use of the tool's customizability and API allowed us to use standard file formats such as XLS and XML as the inputs and outputs to our functional verification requirements tracking flow. If, in the future, a different vendor's simulation environment was chosen we would still be able to make use of the RM tool with minimal changes to the scripts used to extract the coverage information from the coverage database produced by the simulator.

After this flow is rolled out company-wide, focus will shift to tying in other business groups within the company such as mixed-signal IP development, back-end characterization and testing, firmware and software development, etc.

V. REFERENCES

[1] Lee Burns, David Crutchfield, Bob Metzler, and Hithesh Velkooru, "Using Formal Applications to Create Pristine IPs", DVCon 2017.