2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

An efficient requirements-driven and scenario-driven verification flow

Walter Tibboel, NXP Semiconductors, The Netherlands (walter.tibboel@nxp.com)
Heino van Orsouw, NXP Semiconductors, The Netherlands (heino.vanorsouw@nxp.com)
Shuang Han, NXP Semiconductors, The Netherlands (shuang.han@nxp.com)

***Abstract*— For Automotive applications Requirements-Based Verification (RBV) is important. For functional safety applications, the traceability is even more strict to comply with standards such as ISO26262. In the Automotive industry it is common to create test specifications with explicit traceability to functional requirements. Due to the growing amount of detail in the test specification, significant more effort is required in test implementation. For top-level verification with multiple application cores and analog IP, randomization techniques as applied on the IP-level are not effective nor efficient. Therefore, for top-level verification the Metric-Driven and Constraint Random approaches should be combined in a smarter way, by introducing scenario modeling. On top of that, reuse from IP verification into the top-level verification should be improved, both for specification and implementation. This paper introduces a smart combination of directed testing, constrained randomization and scenario-driven verification concepts to improve the verification efficiency and coverage.**

Keywords*— *Verification & Validation; Requirements-Driven verification; Traceability; Metric-Driven Top-level verification; Coverage; Test Specification; Automotive verification

I. INTRODUCTION

A. *Verification approaches*

Functional verification (same for UVM) can be split into a stimuli part (drivers, test sequences, mode settings) and a Monitoring part (pass/fail checks, cover points, assertion results). Of course, both parts have close dependencies to each other. Functional verification is often a mixture of the following Stimuli approaches:

1) Directed test

Known critical cases are easily specified and implemented by directed tests. Directed tests have a clear traceability relation to specific functional requirements (Requirements-based verification). On top-level, directed test have a close relation to use-cases. Directed test is often the preferred way of verifying the correctness of inter-operability of the integrated IP.

2) Constrained Random

With less test implementation effort, multiple combinations are easily explored and covered. Corner cases can be reached that were not specified upfront, which improves the overall coverage. Especially for IP verification this helps to verify many possible combinations, with limited effort. A drawback is result instability [1]: small changes in design or verification can have big impact coverage results.

3) Scenario-driven

Instead of randomization of implementation details, we can also randomize higher-level configuration options in Scenario models. These high-level scenario options have clear relation to the functional requirements. Modeling scenarios in e.g. graphs enable automation through test generation and reporting mechanisms. The high-level scenario models fit well with the verification targets defined in top-level verification.

The relation between the approaches is shown in Figure 1. Since each approach has its own strengths, we need them all.

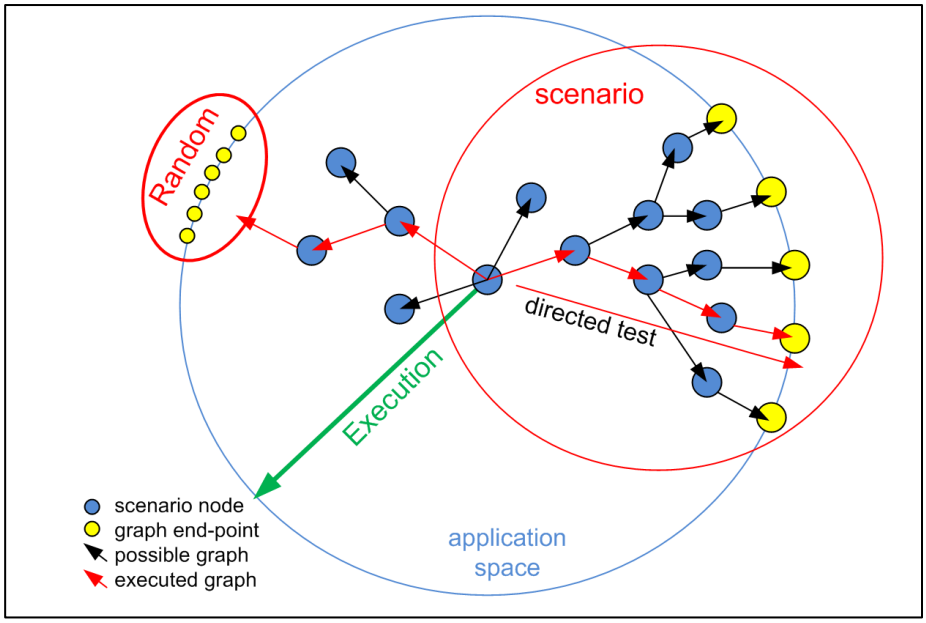

Figure 1 Stimuli approaches covering application space

All three stimuli approaches can be used in combination with a Coverage-Driven (CD) / Metric Driven Verification (MDV) approaches, which focus on the Monitoring part. All monitored results should end-up in one overview, to determine the total coverage.

B. Different roles, disciplines and work environments

The overall chip development has multiple stages involving multiple disciplines. These different disciplines causing different work environments. For this paper, we focus only on two of them:

1) Specification environment

This environment supports all kind of requirements and specifications e.g. related to product, design, test, etc. The environment gives easy access to a broad audience. In the Automotive domain, DOORS is a widely accepted environment for requirements management. Although DOORS offers a rather multi-disciplinary environment, it does not have dedicated features for functional verification, such as coverage and assertions reporting.

2) Functional verification environment

The design and verification engineers are used to work in an IC design environment. This environment often has a verification planning (specification) part and a verification implementation part. The coverage result view has direct connection to all the test details, helping to analyze the results. Cadence vManager is a good example of a tool in which both test can be run, reported and analyzed within the same verification environment.

II. RELATED WORK

Multiple papers have been discussing the differences and benefits of MDV. In [1], coverage-driven is reconciled with RBV, for projects that should comply with ISO26262. It discusses pro and cons of both approaches. It discusses the nature of the traceability between DUT specification towards both approaches. Also the result instability problem of Constraint Random is discussed. Changes in design/verification can have rather unpredictable (coverage) results.

Cadence vManager [2] is positioned in MDV; part of the Functional verification environment. Metrics are split into coverage (cover points), checkers (assertions), and test cases (results obtained by parsing log files). The tool has a planning center. In a GUI-based manner the specs (vPlan) can easily linked to the metric implementations. Metric implementations and results are fully visible in the analysis view. Regressions including parallel job submissions are supported in the regression center. The metric results in vManager are compatible with the Unified Coverage Interoperability Standard (UCIS) [3]. The specification counterpart (vPlan) is based proprietary structure.

DOORS [4] is a requirements management tool of IBM. DOORS supports the OSLC [5] (Open Services for Lifecycle Collaboration), which forms a standardized basis for link management by establishing traceability relations. Relations are defined between different domains. Obviously, Requirements management is one of them. The domain that is most close to specifications for Functional Verification is called Quality management.

In this paper we are proposing improvements specifically for top-level verification by making use of a specific combination of the previously mentioned technologies. The proposed methodology in the paper eases the test specification process to resolve the productivity bottleneck in verification. RBV is considered as one of the key ingredients to facilitate a more rigid flow for Automotive applications.

The TrekSoC solution of Breker [6] enables scenario modeling for multiple verification environments. As the name TrekSoC suggests, it focusses on complex multi-core, multi-threaded, cache coherent system-on-chip (SoC) designs. In this paper we mainly focus on linking requirements with scenario modeling.

III. RBV AND MDV POSITIONED FOR TOP-LEVEL VERIFICATION

Figure 2 presents the verification flow from requirements to coverage, which consists of two distinct areas: the specification and verification environment. The specification environment covers traceability relations from requirement to test items, which are needed for RBV. In the Verification environment three layers are visible, which are closely related to the verification approaches as previously discussed in the introduction section. The figure has a coloring scheme to make horizontal traceability relations explicit: the green color indicates design requirements; the red color shows test specifications, the blue color shows the test implementations, and the yellow color captures the corresponding test results (e.g. outcome of simulations). The light-blue parts show the stimuli, and the dark-blue color represents the monitoring and measurement part. The green and red color cover specification items (the "what"), whereas the blue colors defines the implementation as part of the verification environment (the "how").

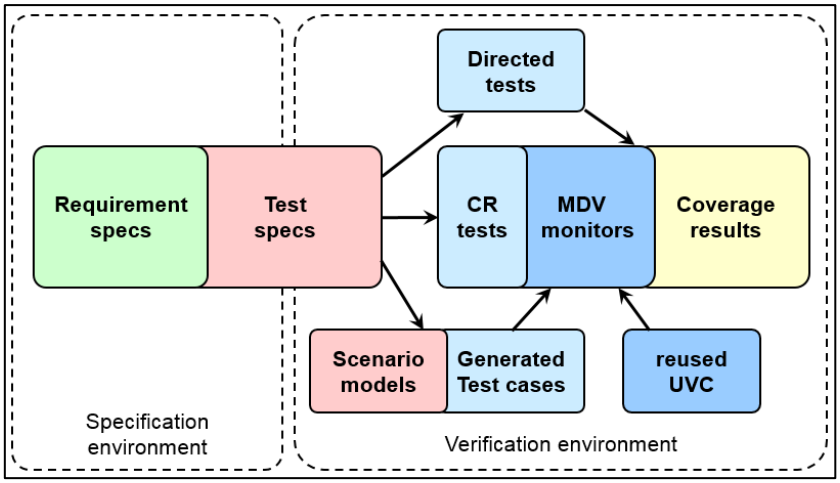

Figure 2 Verification flow from requirements to coverage

Directed tests implement each test item, and each of them reports a pass/fail result. One simulation can cover multiple test items. Test items can be also implemented via a Constraint Random (CR) approach. This approach is close to MDV, because it's fully depending on coverage measured by cover points, checkers/assertions. The Directed test and CR approaches can also overlap, e.g. a directed test with randomization. Or directed test which also contributes to the cover points, part of the MDV approach.

Top-level verification should benefit from the MDV monitors (UVC including checkers and cover points), which are well-matured at IP-level.

For top-level verification, we need both stimuli approaches, but on top of that we need something more. Top-level verification is very naturally expressed in scenarios. The simulations often cover multiple subsystems, each can be configured in a different manner, which will influence each other again. Such decision tree can be easily modeled in a graph e.g., improving insight on the exploration space at top-level. The scenario model is an extension of the test specification (colored red in Figure 2). The configuration options are specified in test items. Other test items can specify a characteristic at the scenario level. Formal modeling like graphs enable to automate test generation. This test case generation can easily explode; constraining the randomness is also needed at this level. Connection to coverage metrics are needed, to monitor what has been executed. Test cases can be implemented as UVM sequences or C-code supporting software-based testing.

IV. PROBLEM STATEMENT

As discussed above, there is a need to improve on productivity and quality. In the following sections the main productivity and quality issues are addressed.

Productivity issues:

- To avoid interpretation errors, test specifications tend to become very detailed, containing a lot of implementation details. With growing design complexity, this becomes labor intensive, the effort of the specification part can easily explode. Maintaining and reviewing will be a huge effort, still not guaranteeing completeness.
- A specification environment has no dedicated support for functional verification. Which means a lot of effort to write the detailed test specifications, and after that a similar effort to "translate" the detailed specifications into tests and a lot of effort to keep both consistent.
- Choices regarding the preferred verification approach (directed test, constrained random or scenario-driven) can be best made in the verification environment. Working out these details in the specification environment is a bit artificial, since in the requirements phase the focus should be on the 'what' and not on the 'how'.
- Limited reuse between IP verification and top-level verification; it's not a matter of copying test specifications and implementations.

Quality issues:

- Because of the high specification effort, often test specification is not complete, and full test-item coverage remains a problem.
- Complete test specification not available early; it should guide the implementation, not describe what has been done.
- Long and detailed specification lists weaken the overview. Designers mechanically implementing the test items and stop thinking about the completeness of this overview.

In the following section a verification flow proposal is presented to address the limitations and issues mentioned above.

V. IMPROVED VERIFICATION FLOW PROPOSAL

In this paper, we propose improvement in two areas. The first improvement is strongly related to better connect the Specification with the Verification environment. The second improvement is discussing how scenario modeling can improve the top-level verification.

A. Improving smooth connection from spec to metric implementation

As described in the problem statement, we experienced work is done twice in two environments. We've been rethinking the traceability relations; what should be specified where. The divide and conquer strategy requires to be refined further.

1) Specification environment

Automotive requirements and test specification can be created and maintained in DOORS. Here full linking and traceability is in place from requirements to test items.

When the requirements are more abstract in terms of features, the test specification can already detail out what needs to be tested to prove the feature. To guide the test specification creation, test items are organized around standardized interfaces (e.g. UART). This will ease UVC and driver reuse in the Verification environment.

The Specification environment is an easy accessible environment for a broader/multi-disciplinary audience. For that reason, no specific support for functional verification is available, the test items are abstracted from MDV specific specifications. The relative abstract test specifications improve the overview (to answer the question: "did we cover all?"), and eases the required early availability. It also improves reviewing by a broader audience.

2) Verification environment

In order to increase efficiency, it is desired that the test specification is detailed with metric goals, which are required for metric implementation. This will decrease rework, interpretation errors and porting effort. In order to get an automated link to the specification environment, these test items are imported from the Specification Environment and become a section headers in what is called the Executable Test Specification (ETS). In Figure 3 the test items are simplified by only TS1, TS2. In practice the test item spec contains multiple attributes describing what should be tested. The term ETS does not mean that the specification itself is executable, but the specification is directly usable for design automation.

Within the Verification Environment there is awareness of functional verification concepts. In the ETS the test items are further refined with specific details on how they should be covered. The test stimuli associated with these ETS items are generated according to the three earlier mentioned stimuli approaches (directed test, constrained random or scenario-driven). The goals for each ETS item are being set that should be met by the MDV monitors. Each test item only passes when all goals inside are met. In case a UVC has been reused with many cover points, the goal heavily depends on the monitor implementations. Reuse of UVC being matured in IP-verification, brings a lot of knowledge and will enrich the ETS refinements.

Refinements of such test item specifications are directly useable for design engineers; specification has become very close to test implementation.

As described in [7], a structured synchronization mechanism and change impact management is required to bridge the Specification and Verification environments. Therefore, this step in the verification flow is a natural location to introduce an abstraction level in the test specification.

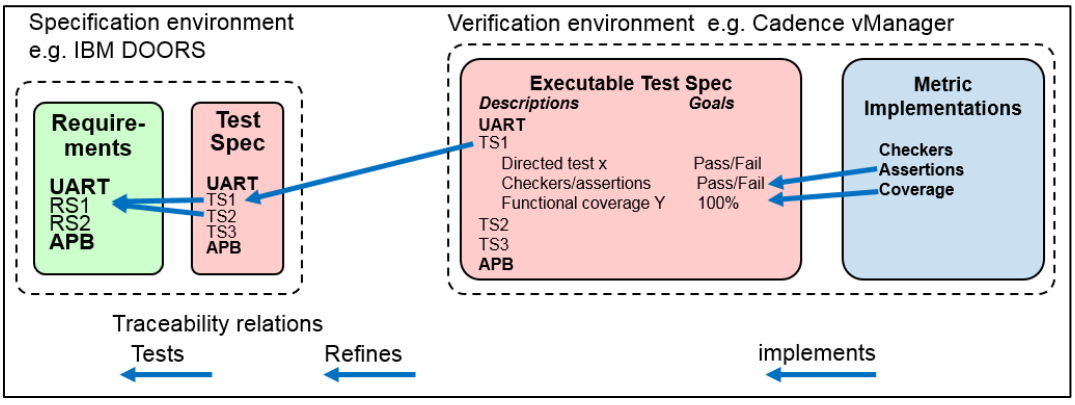

Figure 3 Traceability relations from requirements to test results

Figure 3 shows the different traceability relations:

- Tests: Test items in the test specifications cover the requirements.
- Refines: The ETS refines the test items with functional verification specifics.
- Implements: Checkers, assertions and cover points implement the refined ETS items.

The Verification environment should be able to manage changes from the specification environment (change impact management). Updates in the test items should be clearly flagged. The Verification environment should also ease linking of the refined specifications to the actual monitor implementations. This link causes that simulation results coming from the monitors are directly linked to the refined specifications. Only if all the results are meeting the goals the corresponding test item get the pass result.

B. Scenario modeling improving coverage

In the previous section the focus was on measurement and analysis of the results to see whether the verification goals are met. To trigger the coverage points a set of test cases need to be implemented. A scenario presents all paths on how functional end-points can be reached. This end-point can be linked to a specific verification goal described in a test-item. These scenarios can be defined with an endpoint in mind.

A scenario can be abstract and is created by the system architect early in the design phase. An example of such scenario is given in Figure 4. A test case is a path walking through the graph, as depicted in Figure 5. Arrows starting from an ellipse node should be all followed ("&") in each test case. The diamond node is a choice ("or"), causing multiple test cases. Constrains influence the choices being made during test case generation. At top-level we typically have a hierarchy of graphs, which can cause a huge amount of possible test cases. This graph is the 'blue print' for the verification engineers and indicates what verification IP or drivers/sequencers need to be created to execute the required path. In an abstract view, it makes explicit the relevant choices (configurations) in a scenario.

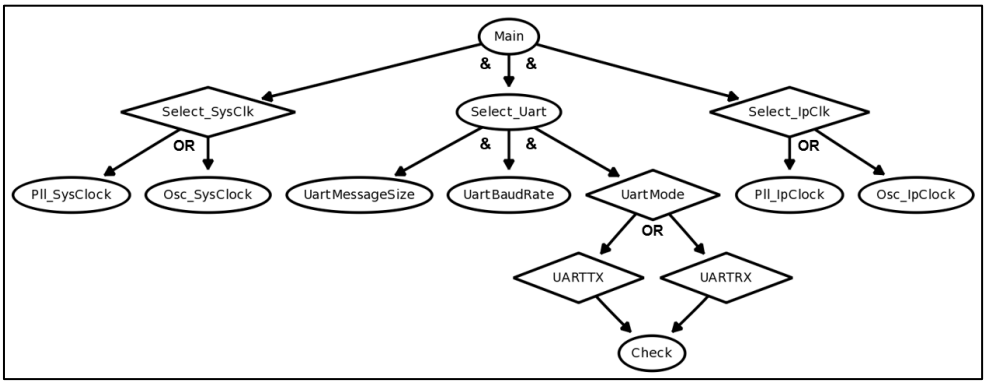

Figure 4 Scenario model from system architect

As the scenario end-points can be linked to the abstract test-items in the test specification, an early indication of the number of paths, and test cases, that need to be implemented is available. One path that defines a test case is shown in Figure 5 below. Automation can be applied in the creation of the test cases. The scenario tool can either create the minimum number of paths to reach the end-points. Next to that with constrains also the path followed can be controlled, resulting in constrained random path creation. Constraints can be used to force certain paths to be followed, resulting in additional test cases reaching the same end-point.

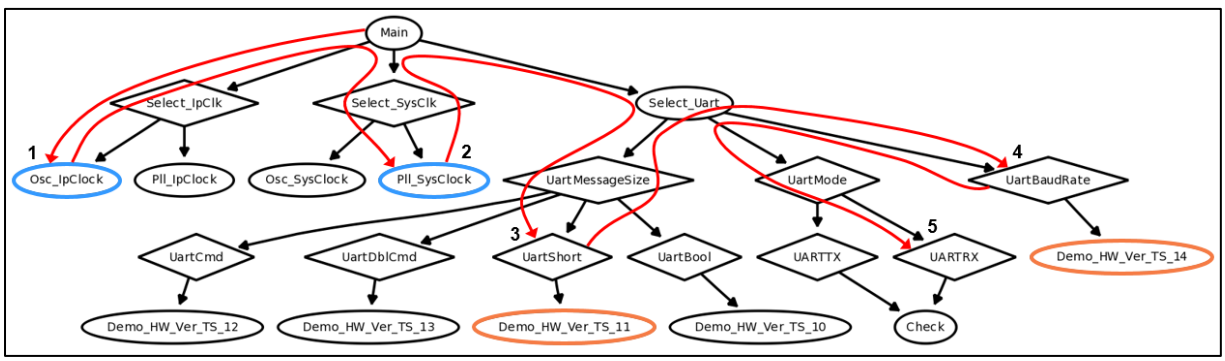

Figure 5 Refined Scenario model from verification lead

E.g. in figure 5, a combination of Osc and Pll clock is used. If the system is required to start with only Osc clock, then a constraint can be added (IpClk=Osc && SysClk=Pll), resulting in different path to same endpoint. This analysis can be done before any of the detailed test case code is implemented.

Once the paths are determined, the nodes can be refined with code to be executed during the simulations. The code can be e.g. C-code to be executed on an embedded processor, or code for a UVM test (or even both). When the implemented test cases are executed, metrics monitored by the UVC's, give a view on the quality of coverage.

When the coverage in the ETS shows that not all detailed goals are met, the graphs from the scenarios can be reviewed to see if it is complete or if the graph is corrupted by another unexpected defect. Or additional constraints can be set for the generation of different or additional paths. This coverage feedback closes the loop of test case generation to covering goals. The generation of additional paths is low effort as the scenario in it completeness was already implemented.

In our demonstrator, we have implement change impact management and traceability linking from the test item specifications into the scenario sources, using technology discussed in [7]. Changes in the test specifications will have impact on the scenario model, causing changes or refinements in the scenario model.

VI. DEMONSTRATOR RESULTS

In the demonstrator, a UART has been used as the IP, for this a Test Specification and Executable TS has been generated. A reused UVC and its vplan has been integrated. In the ETS specific goals are set to what extend each item must be covered. These goals are merged to get an overall result per test item depending on different coverage types:

- Directed tests are covered by using an assertion for a specific end-state.
- Randomization is introduced to collect how many options of a setting are used.
- Scenarios are used to constrain the number of test cases that are required to cover the specified test items.

The results of a regression are shown in the Figure 6. Here the constrained scenario ends up in the execution of three test cases. A test case is the implementation of one or more test items. The coverage of these test cases is well visualized in the vManger/vPlan environment.

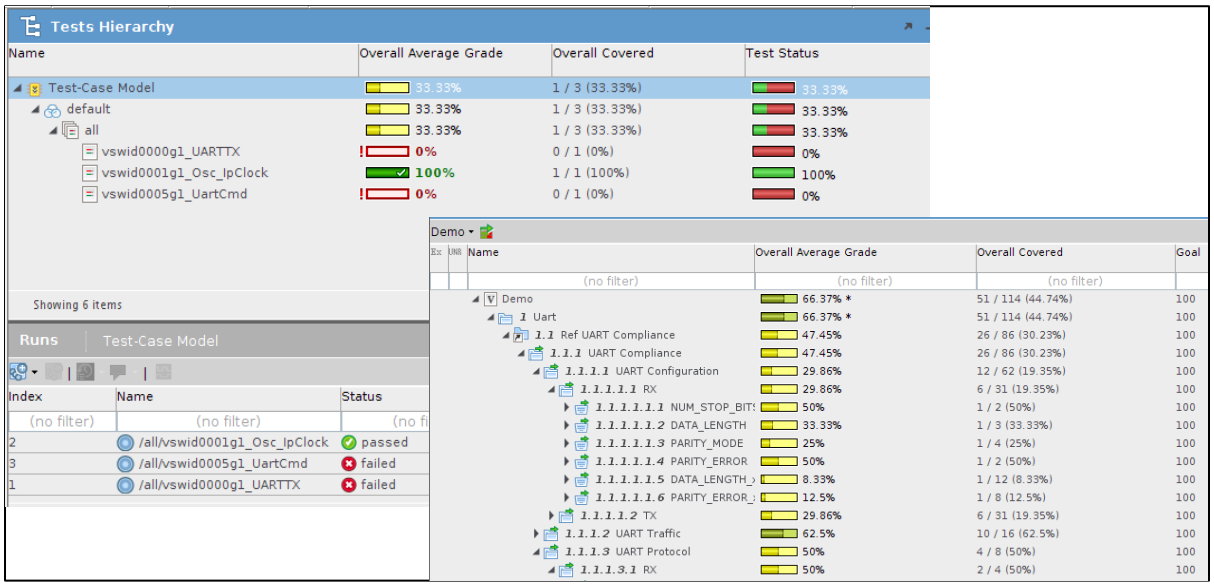

Figure 6 Results of one scenario and corresponding UART coverage results

VII. CONCLUSIONS

For Automotive application, requirements and test specifications are created as the starting of verification process (RBV). Traceability links are maintained from requirements to the test specifications. Furthermore, a structured abstraction hierarchy is essential, to keep the specification effort reasonable for coming designs.

Different verification approaches, such as directed test, constrained random or scenario-driven, are used as part of the test implementation. However, application of these different concepts is often not well connected to the test specification phase. Therefore, the ETS (Executable Test Specification) proposed in this paper is introduced to refine test specifications for the verification environment. This approach improves specifications to be early available in the proper format for the verification engineers.

In addition, scenario modeling is introduced to increase the coverage and confidence of test completion. For top-level verification, these scenario models act as 'blue print' for test case creation, and smoothen the handover from architect to verification engineers.

Reuse from IP verification to Top-level verification can be best achieved by passive UVC. Structuring test specifications around (standard) encourages the reuse.

The proposed methodology has been applied successfully on a UART subsystem, and showed that a smart combination of directed testing, constrained randomization and scenario-driven verification concepts improves the verification efficiency and strengthening the overall test coverage.

ACKNOWLEDGMENT

Special thanks to NXP colleagues, to Monica Farkash for helping with scenario modeling and generation, to Martin Barnasconi and Erwin de Kock for their contributions in discussions. Thanks to Cadence R&D and in special to Thomas Ziller for the vManager support.

REFERENCES

[1] Avidan Efody, "Who takes the driver seat for ISO 26262 and DO 254 verification? Reconciling coverage driven verification with requirement based verification", DVCon-Europe 2015

[2] Cadence® vManager™ Metric-Driven Signoff Platform http://www.cadence.com/content/cadence-www/global/en_US/home/tools/system-design-and-verification/planning-and-management/incisive-vmanager-solution.html

[3] UCIS, Unified Coverage Interoperability Standard, http://www.accellera.org/downloads/standards/ucis

[4] IBM® Rational® DOORS®, requirements management supporting traceability and scalability, http://www-03.ibm.com/software/products/en/ratidoor

[5] OSLC, Open Services for Lifecycle Collaboration, http://open-services.net

[6] Breker, TrekSoC http://www.brekersystems.com/products/treksoc

[7] Walter Tibboel, Jan Vink, "Closing the loop from requirements management to verification execution for automotive applications", DVCon Europe 2015