Enhancements of metric driven verification for the ISO26262

Recent advancements better supporting
specific verification requirements including functional safety

Michael Rohleder, Clemens Röttgermann, Stephan Rüttiger
AMP/New Product Development Center
NXP Semiconductors, Munich, Germany
{michael.rohleder,clemens.roettgermann,stephan.ruettiger}@nxp.com

John Brennan, Matt Graham,Riccardo Oddone
Advanced Verification Systems
Cadence Design Systems, San Jose, California, USA, and Milan, Italy
{brennanj,magraham,ric}@cadence.com

***Abstract*—advanced methods are becoming a more and more popular choice, enabling increased quality and productivity by selecting the most suitable approach of the verification work. Metric driven verification has suffered from the need to combine the results of different methods, related limitations have been lifted by recent developments. The impact of these changes for the verification of functional safety devices are identified and further discussed.**

***Keywords*— SoC, functional safety, ISO26262, traceability, formal verification, simulation, requirements, metric driven**

I. INTRODUCTION

Functional verification is an essential part for any semiconductor development; which involves many disciplines and uses different methodologies to ensure the implementation matches the design intent. Related tools and methodologies are constantly evolving to counter the continuously growing complexity and performance requirements. One particular methodology that has been proven to be very powerful is the usage of formal methods; especially property checking and assertions for this purpose. Having the ability to select the most suitable verification method for a particular problem provides more flexibility and results in a higher productivity of the overall verification.

An environment that uses multiple approaches for the verification of a semiconductor device puts some additional burden; especially when it is required to proof an appropriate verification coverage has been reached. The usage of multiple methodologies requires to combine the reached coverage of any of the used methodologies, which is often a complexity on its own. Having appropriate tool support for this task can become an essential feature for devices that are intended to be applied to safety-related systems, which are often specified in the context of Functional Safety standards, like the IEC61508 [4] or its descendant for Automotive Electric/Electronic (E/E) Systems, the ISO26262 [3]. This paper reflects on interdependencies of improved tool support and standard requirements for this particular problem.

II. THE ISO26262 STANDARD

Functional safety is a concept applicable across all industry sectors that is fundamental to the enabling of complex technology used for safety-related systems [1] [2]. It is the part of the overall safety of a system or piece of equipment that depends on the system or equipment operating correctly in response to its inputs, including the safe management of likely operator errors, hardware failures and environmental changes. Its objective is the freedom from unacceptable risk of physical injury or of damage to the health of people either directly or indirectly (through damage to property or to the environment). As such it is fundamental to the enabling of complex technology used for safety-related systems by providing the assurance that these systems will offer the necessary risk reduction required to achieve safety for the equipment.

Functional safety is intrinsically end-to-end in scope in that it has to treat the function of a component or subsystem as part of the function of the whole system. Related efforts form an integral part of each automotive product development phase, ranging from the specification, to design, implementation, integration, verification, validation, and production release. To cope with this complexity, the ISO26262 standard consists of 9 normative parts and a guideline as 10th part, which defines functional safety for automotive equipment. Like its parent standard IEC61508, the ISO26262 is a risk-based safety standard, where the risk of hazardous operational situations is qualitatively assessed. It addresses possible malfunctions by defining safety measures to avoid or control systematic failures and to detect or control random hardware failures, or mitigate their effects. It does not address the nominal performance of E/E systems. In particular, the ISO26262:

- provides an automotive safety lifecycle and supports tailoring the necessary activities during its phases.
- covers functional safety aspects of the entire development process.
- provides an automotive-specific risk-based approach for determining risk classes (ASIL=Automotive Safety Integrity Level).
- uses ASILs for specifying the item's safety requirements for achieving an acceptable residual risk.
- provides requirements to ensure a sufficient and acceptable level of safety is being achieved.

Of particular interest for the following discussion is the required usage of a tool or tool-chain for requirements traceability, which intends to minimize the amount of manual requirements management. A second notable influence of the ISO26262 standard is the relation to a state-of-the-art implementation, which intends to ensure the usage of appropriate tools and methodologies. Of particular importance is the fact that a requested approach does not get outdated when recent developments result in improvements that can but also should be exploited in the context of such a standard.

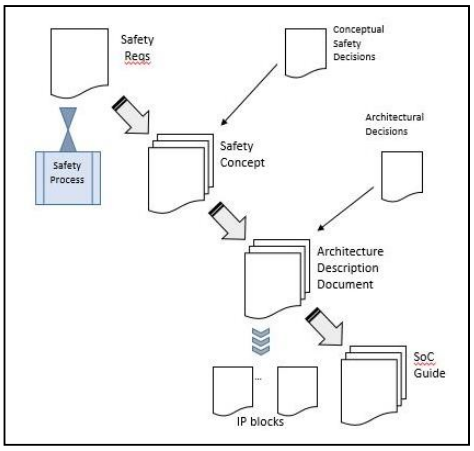
III. SAFETY REQUIREMENTS AND REQUIREMENTS TRACEABILTILY

In product development and process optimization, a **requirement** is a singular documented physical and functional need that a particular design, product or process must be able to perform. It is a statement that identifies a necessary attribute, capability, characteristic, or quality of a system. A specification may refer to an explicit set of requirements to be satisfied by a material, design, product, or service. In the classical engineering approach, sets of requirements are used as inputs into the design stages of product development. Requirements show what elements and functions are necessary for the particular project; they are also an important input into the verification process, since tests should trace back to specific requirements.

The ISO26262 standard notes that **safety requirements** can be specified using natural language (usually recommended for higher level safety requirements) or using informal, semi-formal (strongly recommended for lower level safety requirements and higher ASIL levels), or even formal notations. In particular, the ISO26262 standard imposes some attributes and some characteristics and properties on safety requirements, as this is depicted in the figure at the right side, which illustrates those “requirements on safety requirements” as they are imposed by part 8 of the ISO26262 standard. Another need imposed by the ISO26262 on safety requirements is the placement under configuration management.Requirements usually come from different sources, reflecting the many different people or groups having specific needs on a product. What matters in requirements management is a structural evolution: a trace of where requirements are derived from, how they are satisfied, how they are tested, and what impact will result if they are changed.

Requirements traceability is a sub-discipline of requirements management that is concerned with documenting these relationships. It allows to document the transformation of higher level requirements into successively more concrete (lower level) requirements and other development artifacts; thus describing the relationships between layers of information. This provides the ability to bi-directional define, capture and follow the traces left by requirements on other development artifacts (requirements, specification statements, designs, test cases, test procedures, test results, models and developed components) and the trace left by those elements on requirements; thus enabling to trace back an implemented feature to its originating requirement. This can be used during the development process to prioritize requirements, or to determine how valuable a particular requirement is for a specific user or group or purpose. In particular, the ISO26262 standard specifies that safety requirements shall be traceable with a reference being made to:

- each source of a safety requirement at the upper hierarchical level
- each derived safety requirement at a lower hierarchical level (or to its realization in the design)
- the specification of requirements verification, selecting the methods to be used (e.g. review, analysis checklists, simulation scenarios, test cases/data/objects).



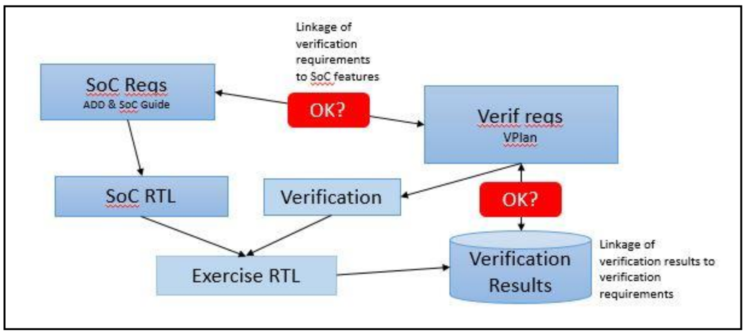
The picture at the right side depicts conceptually a possible breakdown of safety requirements into derived requirements, their mapping onto SoC features and specific features implemented by a particular IP block. It also shows exemplary the involved documentation, identifying the documentation to be included in a traceability tree.

In order to support the management of safety requirements, the use of suitable requirements management tools is recommended by the ISO26262 standard. This will allow completeness checks, the assessment of requirements deviations over all levels, and a tools supported presentation of the qualification status.

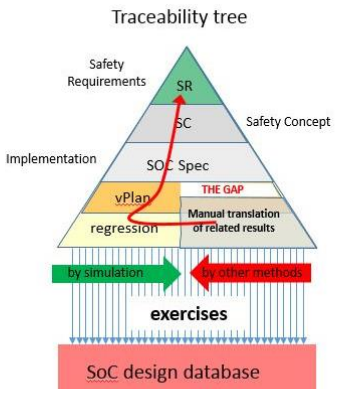
IV. TRACEABILITY OF SOC FEATURES TO VERIFICATION RESULTS

As already noted, the ISO26262 standard requires the traceability of safety requirements down to the actual implementation by development artifacts and the verification of these artifacts.

The mapping of safety requirements to features implemented by the SoC architecture or a particular IP block (as shown left) enables the usage of existing verification ecosystems for this purpose. Tagging of SoC and IP block features and determining the verification coverage for these features is a well-known and widely used methodology in the semiconductor industry. As such having the ability to re-use the infrastructure that is already available for determining the verification coverage provides a significant incentive.


Linkage of verification requirements to SoC features
SoC Reqs
ADD & SoC Guide
OK?
Verif reqs
VPlan
SoC RTL
Verification
OK?
Exercise RTL
Verification Results
Linkage of verification results to verification requirements

As such, combining verification coverage metrics with the need to trace safety requirements, their implementation and verification allows to use a mature and existing infrastructure for this purpose, which avoids the usage of additional tools or the creation of further flows that are supporting the related requirements. Unfortunately this is only a valid statement for the traditional simulation based approaches; as soon as there are other methodologies involved, the creation of the required coverage metrics and the tracing of the results becomes significantly more problematic.



The figure at the right side depicts a potential solution for this problem, here the verification results determined by other methods than simulation are derived into simulation results, which are then picked up and further processed by the regression tool. Of course such a manual translation is actually not the most appropriate way of a tool supported feature tracing, and actually creates a gap between the SoC specification and the verification of related features. As a result a more seamless inclusion of verification results and associated coverage information into the requirements traceability flow is desired and can result in some significant improvement of corresponding productivity and the quality of related work products. As such, having this capability redefines the state-of-the art for SoC verification, when a combination of simulation based approaches is used with other methodologies.

V. Evolution of Verification Methods on the SoC Level

The usage of all those methods at the IP and sub-system level, including metric driven verification (MDV) and constrained random stimulus generation, apply equally at the SoC level, but in slightly different scope. Additionally, all these methods must be further focused and refined, owing primarily to the increased complexity and solution space of the verification problem at the SoC level. Furthermore, in the case of RTL simulation, the concept of use case scenarios can be introduced, further refining constrained random stimulus generation. This also allows for further abstraction of the functional coverage to be collected to include driver, firmware, OS and even application layer use cases, which might result in more suitable, but also more complex coverage data to be collected.

Formal methods utilized at the SoC level must be similarly focused. One such example is the application of formal to check for connectivity issues. In this case, the formal rules and assumptions take the form of descriptions of connectivity between various IP blocks within the system. Formal methods can then be utilized to definitively prove that the desired connectivity and the implemented connectivity match. Other methods may then be used to verify other aspects of a particular feature, and only the combination of those checks finally results a feature to be completely verified.

The focusing of these verification tasks, along with integration of others (analog mixed signal simulations, test bench acceleration, emulation, virtual platform, HW/SW co-verification, etc) means that the types of metrics to be collected, aggregated and considered are now being generated by a number of different engines. It also means that it is no longer viable to verify and re-verify each piece of functionality at every level of integration with the same methodology, but usually a combination of those methods is required. Additionally, the usage of involved verification approaches may vary on a per feature bases, in some cases even multiple approaches may be needed to completely and consistently verify a single feature. Further, the volume of data has increased by perhaps orders of magnitude. As such, the MDV solution requires a number of capabilities for application to the SoC:

1. Collection, aggregation, and analysis of metrics from a variety of engines. Data from disparate sources must be analyzed, so that a true picture of status across the projects execution engines can be easily reviewed.

2. Merging of metrics from varying scopes of verification. Data collected at the IP, subsystem and SoC level must merge at the appropriate points to enable taking credit for verification activity at all levels of integration.

3. True re-use of verification plans. Cut-and-paste re-use is not sufficient for large plans that are aggregated together. Verification plans must be able to be parameterized and instantiated within higher level plans.

4. Capacity and scalability. The ability to aggregate data across varying levels of integration and varying engines will not be useful if the collection and analysis engines cannot scale with the amount of data produced.

VI. COVERAGE TYPES, PROS AND CONS

Test coverage, code coverage, assertion coverage, functional coverage, and fault coverage are the primary metric types which is used in the combined functional verification and function safety process. These metrics are the technical metrics which define to what extent a feature or function has been tested. Additionally, there are some management metrics which are extremely useful as well, such as planned coverage and requirements coverage, both of which would be considered higher level completion and status grades to the overall project or program.

As was discussed, metrics are used for different purposes and will have different and sometimes overlapping meanings. Generally however we wish to use metrics to prove that a particular requirement has been met, and for such the following steps are required:

1) Specify the complete list of involved tests within the regression suite.
2) Identify the correlation of the selected tests to the functionality defined within the specification.
3) Prove that the test executed the functionality, it is supposed to check.
4) Create a list showing each function and check off those that were exercised.
5) Define an overall coverage grade and the contribution of a particular test to this grade.

The primary objective in pre-silicon functional testing is to prove the functionality of the design. Code coverage, knowing how much of the code was exercised by the design, cannot tell you this information, and therefore is a nice complementary metric, but secondary to functional metrics. Code coverage is not used as a signoff metric, but more as a tool to determine how much additional test development may be needed. Complete execution of each test in a test suite can be a measure functional coverage. Each test is created to check the particular functionality of a specification or requirement. Therefore, it is natural to assume that if it were proven that each test is completed properly, then the entire set of functionality is verified. While this is true, tests which pass or fail is a binary measure of success, and normally not to the granularity needed to satisfy functional requirements. Functional coverage is the determination of how much functionality of the design has been exercised by the verification environment and can be rolled up into a coverage grade. By using functional coverage, the process of checking the functionality can be more precise and also automated, which is a necessity for large designs.

The table below shows the various types of metrics which may be used in a ISO26262 project.

Metric	Type	Use
Code Coverage	Code	Determine how much of the design has been tested
Functional Coverage	Functional	Use Cover Group to test functions compared to their specification
Assertion Coverage	Functional	Easy method for designer to check for properties in the design
Test Coverage	Functional	Amount of tests in a regression which are passing or failing
Fault Coverage	Safety	Stuck at and Transient Fault Coverage for safety classification
Requirements Coverage	Management	Percentage of requirements or specifications which are completed
Plan Coverage	Management	Defines what amount of the plan is implemented in actual tests

Traditionally all of the metric types shown above are derived from the simulation environment. As these metrics and coverage types become an essential element of signoff, alternative methods have been created to more quickly get the results. Formal technology for example, operates on formal proof engines which operate substantially faster on portions of the verification problem. Formal application such as connectivity checking, or register verification, are plug in style applications which automate the verification of connectivity and registers which out having to write new tests. Key to the success of using alternate methods is the ability to bring the metrics and coverage data and combine it with the other tests. Assertions for example, can run statically in a formal environment, or dynamically in a simulation environment, and both can be used to collect verification results and seamlessly combine them. Assertions are often called checks, but can also be considered to be a coverage item and useful in functional verification. For automotive and ISO26262 environments, measuring the functionality with functional coverage and have traceability of such tests, is the most significant thing.

While functional metrics become the critical one, there are advantages and disadvantages of using this technique; specifically, 100% functional coverage indicates that all items in the requirements list have been tested, but there are advantages:

- Functional coverage helps determine how much of your specification was covered.
- Functional coverage gives feedback about the untested features.
- Functional coverage provides the information about the redundant tests which consume valuable cycles
- Functional coverage enables cross’s of functions for detailed hole analysis (what is not covered)
- Functional coverage can be represented as a single metric grade, easy to understand

On the other side, there are some disadvantages that must be taken into account:

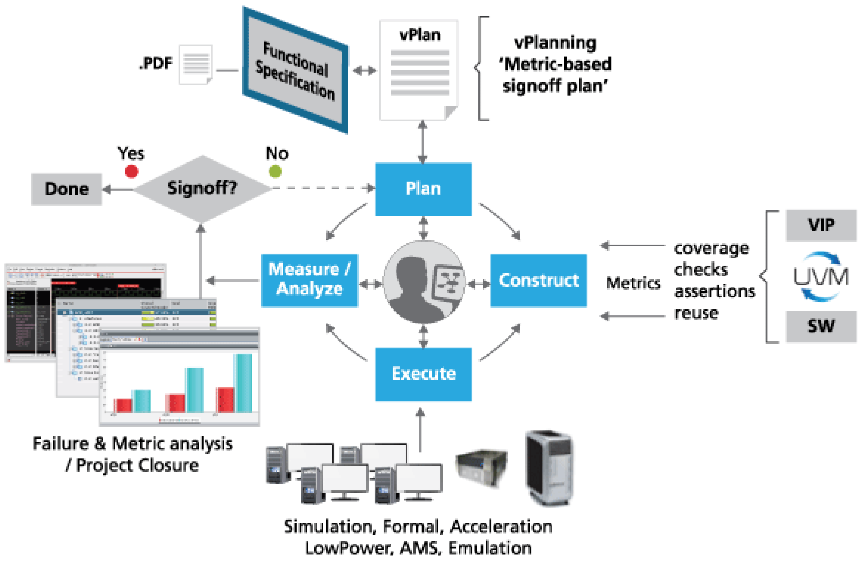
- There is no real way to check that the coverage model is correct, manual check is the only way
- With functional coverage there is a possibility of still not exercising some parts of the HDL code
- Functional coverage cannot identify unreachable states or dead code

For each of the above disadvantages, there are also now well proven solutions. The coverage model size, while a manual process, can be determined by comprehensive verification planning. The missing parts of the HDL code can be seen thru hole analysis and thru constraint adjustments, and lastly unreachable and dead code can be automatically found using formal technology, and automatically excluded or waived from the verification plan..

VII. FEATURE COVERAGE BY VERIFICATION

There is often confusion about the definition of what constitutes a verification plan and what constitutes a test plan. The definition should be clear: a test plan defines a fully elaborated list of tests that will be created and the completion of this list defines completion of verification. A test plan alone, however, tends to be a poor mechanism as it already contains decisions about the execution of the process. In contrast, a verification plan captures “what” that needs to be verified but does not define the execution—the “how.” This is a crucially important distinction, as it is allows or even expects that verification may be done using multiple technologies by multiple specialists. The “what” of the verification plan should be referred to as the “features” or “goals” that need to be verified—it is the design intention. When planning a project, the engineer should not presume the underlying tool by which a feature is verified. It is likely that execution of the verification effort will come from several sources, and the results will be represented by various forms of coverage, as described earlier.

As such, the creation of a verification plan is an iterative process, involving different verification methods, but also applying learning and refinement during each cycle to further improve the result. This process is depicted in the following diagram.



There are two significant benefits to feature based verification management. First, features are easier to understand and communicate between technical and non-technical people. Operating at a feature level means managing at a higher level of abstraction, important for improving verification productivity. Second, features are easily connected to device specifications and requirements, enabling a direct, and clearly traceable path from requirements, to implementation, and finally to verification.

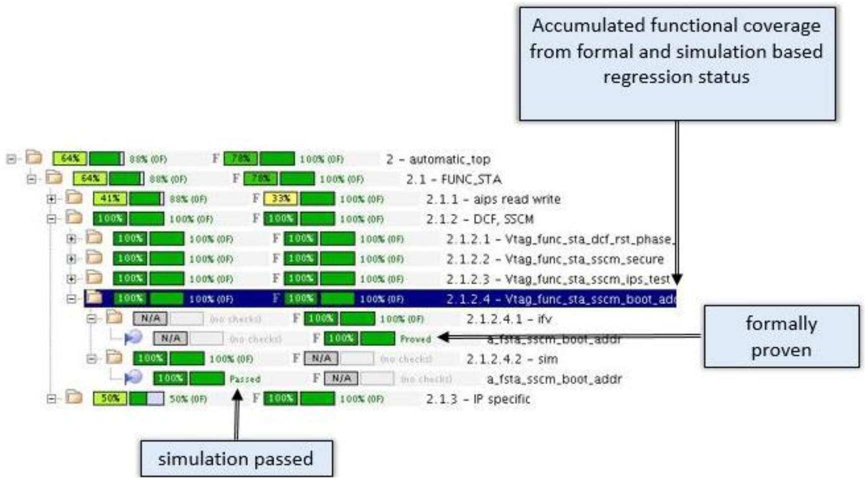
During the verification planning process feature based orientation should begin to be used. When creating the verification plan, it is important and useful to identify abstract features and hierarchies of features that closely resemble the hierarchy of the specification and requirements. While not mandatory, a verification plan is often a useful convenience that helps designers and verification engineers communicate. Verification plans can be hierarchical and integrate other (sometimes commercially supplied) verification plans. This mechanism enables reuse of plans that relate to standard interfaces or perhaps blocks of IP that are reused across multiple projects. It can also be useful for segregating information as from block level to system level, or at different milestones where different goals are required at various development stages. A fully assembled verification plan connects the requirements and specifications to the abstract features of the device. This enables complete traceability between the device implementation and the defined requirements. The following section will describe connecting the collected metrics (e.g. coverage) to the Verification Plan to further enhance traceability from requirements through feature verification.

VIII. TOOL SUPPORT FOR FEATURE COVERAGE AND COLLECTION OF COVERAGE RESULTS

Defining coverage metrics to be collected, and specifying which coverage metrics relate to which features or requirements to be verified achieves a great deal in terms of completeness of the verification effort. However, truly capturing the completeness of a verification effort requires analysis of the results of the execution of tests and the collection of various forms of coverage against that plan. This is effectively annotating simulation, formal, emulation and other results onto the verification plan, making the plan itself a living, executable document that represents the sign off criteria and contract for the device under verification (DUV).

To enable creation of executable verification plans, engineers must be able to connect implementations of tests and coverage to various sections of the hierarchical plan. While this has been accomplished in the past with various combinations of scripting, parsing results logs, and spreadsheets, these methods have proven inefficient and cumbersome, resulting in poor, incomplete or even ignored verification plans. Capturing verification plans requires specifically tailored applications, with capability to both enter, edit and re-organize hierarchical plans, as well as visibility to the metrics model. The unification of the plan creation and metrics model in a single application enables point and click mapping of metrics to the verification plan, as well as reporting on which portions of the verification plan may be incomplete. By using a specifically tailored application for verification plan capture, not only is the capture and edit of the verification plan itself more efficient, but the connection to the metrics is faster and more accurate. The resulting verification plan is more complete, and as a living, executable document, able to be used consistently for the primary (if not sole) measuring stick for verification sign off. The resulting increase in visibility to the verification process has been proven time and again to result in more efficient, more effective, and more complete verification.

To this point, verification metrics of various types have been discussed, their various merits and uses described, and the importance of their mapping to an executable verification plan defined. Another important aspect is the collection and grading of the corresponding verification results. By defining the verification plan, and the required tests and coverage to fully verify the DUV, the "how" portion of the verification project can be determined. It is critical that up to this point that the verification plan does not dictate the execution engine to be used for a specific feature. Formal methods, both automatic and expert, digital logic simulation, analog mixed-signal simulation, acceleration and emulation must all be available for the verification engineer to utilize. The application for capturing the verification plans and annotating the results must be aware of these various verification engines, and able to collect their results and annotate the results equally. The following figure identifies a simple example, where a feature is checked by a combination of simulation based tests and some formal proofs.



64%	88% (OF)	F	78%	100% (OF)	2 - automatic_top
64%	88% (OF)	F	78%	100% (OF)	2.1 - FUNC_STA
41%	88% (OF)	F	33%	100% (OF)	2.1.1 - aips read write
100%	100% (OF)	F	100%	100% (OF)	2.1.2 - DCF, SSCM
100%	100% (OF)	F	100%	100% (OF)	2.1.2.1 - Vtag_func_sta_dcf_rst_phase_
100%	100% (OF)	F	100%	100% (OF)	2.1.2.2 - Vtag_func_sta_sscm_secure
100%	100% (OF)	F	100%	100% (OF)	2.1.2.3 - Vtag_func_sta_sscm_ips_test
100%	100% (OF)	F	100%	100% (OF)	2.1.2.4 - Vtag_func_sta_sscm_boot_add
N/A	(no checks)	F	100%	100% (OF)	2.1.2.4.1 - ifv
N/A	(no checks)	F	100%	Proved	a_fsta_sscm_boot_addr
100%	100% (OF)	F	N/A	(no checks)	2.1.2.4.2 - sim
100%	Passed	F	N/A	(no checks)	a_fsta_sscm_boot_addr
50%	50% (OF)	F	100%	100% (OF)	2.1.3 - IP specific

Accumulated functional coverage from formal and simulation based regression status
formally proven
simulation passed

At the verification environment implementation stage, be it a simulation environment with UVM, a formal assertion based environment, or an environment using some other execution engine, the collection of metrics remains the same. Tests still pass and fail, coverage information is generated or collected to determine which abstract features or behaviors were stimulated. The verification plan must be able to be annotated with results from all of these environments. Further, the verification engineer must be able to define cases in which one or more specific verification engines must cover the same feature, or other cases where any one of the various engines at their disposal is suitable for collecting coverage. A verification management application must allow for regression results to be collected, merged, and reported per the hierarchical format of the verification plan. This enables an objective measurement of the verification progress, organized per features and requirements, with the ability to extract both high level summary information, and low level detail.

Consider the verifying an IP such as a bus arbiter of some kind. The verification engineer can, and should, choose the most effective environments for verification. In this case, that may be both a simulation based UVM environment AND a formal assertion based environment. In such a case, a number of assertions can formally prove in the formal environment, and can also be used as sequential checks in the simulation environment. Additionally, further metrics from both environments, some of which are redundant and some of which are complementary will be generated. The verification engineer must be able to annotate certain sections of the verification plan in which an assertion must be verified in BOTH the formal and simulation environments. Similarly, certain section of the verification plan could require that the assertion be verified in either environment, so long as it is verified at all.

Clearly this concept of collecting and analyzing and reporting metrics can extend to any of the other verification engines mentioned above, as well as other complementary verification metrics. These capabilities result in the following requirements of the verification engines:

1. A common verification metric (coverage) collection and storage format across verification engines.
2. Common languages and standards across adjacent verification approaches (e.g. SVA assertions to define rules for formal analysis and sequential properties in simulation)
3. The ability to aggregate or merge the results across engines

Knowledge by the planning and management tools of which engines produced which results.

IX. Fault Simulation

Fault simulation is evaluating the effect of faults on the behavior of a device, which is of special importance for devices targeting an application in the functional safety domain. Verifying the appropriate behavior is required to ensure the correct and complete implementation of corresponding detection features often implemented within corresponding devices. This is a completely new area of concern that is specific for functional safety, because classical verification covers only behavior *in the absence of failures*; which might occur instantly and arbitrarily everywhere within a device (e.g. caused by an alpha particle, radiation, or another environment effect). The need to also account for failures is usually reflected in current verification approaches by specific directed tests that inject faults at a particular point within the device to exercise the result effects; usually by applying forces.

Recent developments in this area are providing some substantial improvements for this topic; by supplying a specific methodology for injecting faults that can be better observed and easier distinguished from normal behavior. Additionally, there is now tool support for the generation and evaluation of the associated fault universe; a functional safety term that relates to the set of faults that may occur within a certain block or subsystem of a device.

X. Results, Summary and Outlook

With the advent of more recent verification methodologies, the requirement to provide evidence for achieving an appropriate verification coverage – which is imposed by the functional safety standard ISO26262 – became an issue; the combination of corresponding results from multiple tools that are using different approaches caused issues that could only be solved with manual effort for some time. This paper identifies how recent developments enable welcome enhancements to combine results from simulation based verification with results that stem from a formal verification run. The effect of such enhancements in the context of the strict requirements imposed by the ISO2626 have been detailed; covering also the corresponding impact on the need for tool based traceability.

Unfortunately, especially for functional safety devices there is already new harm on the horizon. The need to also cover the verification of behavior in the context of injected faults has resulted in some important enhancements that provide basic support for related verification efforts. Like any new verification methodology, such advancements provide a new set of issues for any MDV methodology, since the resulting data must be combined with other verification results. The specification of coverage metrics in this context is a wicked problem, since there is a wide range of potential failure types, opening the space for further research and development in this area.

References

[1] http://www.iec.ch/functionalsafety

[2] Portions of the following text are citing statements taken from http://en.wikipedia.org/wiki about: Functional_Safety, IEC_61508, ISO_26262

[3] www.iso.org

[4] http://www.iec.ch/functionalsafety/standards