STRUCTURED ASSERTION DESIGN VERIFICATION FOR COMPLEX SAFETY-CRITICAL HARDWARE

Kristoffer Karlsson, Contractor

Håkan Forsberg, Saab Avitronics, Jönköping, Sweden

Abstract

Previously we have proposed an approach to meet the certification requirements for the highest design assurance levels for complex safety-critical hardware in avionics. We named this the *Overlapped Layered Modular Methodology* (OLMM), which involved the use of assertions for both *Formal Verification* (FV) and simulation-based verification.

One of the biggest obstacles in using assertions for functional design verification is the essence of asserting design functionality. To precisely specify the functionality of the design and capture it fully in a formal yet understandable expression, that is also simple enough to be computable in a formal proof calculation. In addition to this, and maybe even more important within the area of complex safety-critical design, to achieve confidence that these goals have been met.

To support functional design verification with assertions using a methodology such as the OLMM we propose the use of UML modeling with functional test trees to confine verification efforts to specific functions and to direct the assertion capture process. We believe that using this structured process to formally specify design functionality will contribute to a higher confidence and visibility of verification means and results of complex designs, which in turn leads to higher design assurance.

Introduction

The certification guidelines *RTCA/DO-254* (DO-254) [1], requires additional verification efforts for the highest design assurance levels (A and B) to assure that complex electronic hardware designs behave as intended. Since DO-254 was accepted the avionics industry has struggled with interpreting these guidelines and developing processes to cope with this standard and the additional verification methods suggested.

In previous studies we formulated a DO-254 compliant design assurance strategy for complex level A and B hardware designs. As mentioned above we called this the *Overlapped Layered Modular Methodology* (OLMM) [2]. The OLMM involved the use of formal functional specification with PSL[1] assertions in addition to traditional HDL test bench simulation. With the use of assertions more expressive and efficient means of describing test cases was made possible and it also enabled the use of *Formal Verification* (FV) for in-depth proof calculation.

The OLMM was also evaluated in a technical demonstrator [3], a design highly applicable for use in avionics, bridging the widely used ARINC 429 bus interface. This evaluation was made with a variety of verification tools and techniques available on the market. In this study the OLMM proved to be purposeful and the experience gained was very constructive for the continuing development and improvement of a DO-254 compliant process.

Since this evaluation was made (in 2006) a couple of interesting things have happened on the market. As there has been a growing interest from the avionics industry in advanced verification methods and certifiable circuits the tool vendors and semiconductor manufacturers themselves have taken an interest in the industry's struggle with compliance to DO-254 [5] [6] [7]. We have therefore seen an increased flow of means and methods to fulfill certification objectives.

Another development is that formal assertions have during the last couple of years become widely used in HDL verification, and also within the avionics industry and the area of safety-critical

[1] *Property Specification Language* [4]

design. However, even if our experience shows that the use of formal specification and verification of design functionality is of great use there are some fundamental shortcomings with this approach that needs to be addressed to assure a well-organized implementation and to increase the value of results acquired with using assertions.

Our previous experience using formal specification/verification has shown that a large amount of assertions makes it hard to reason about the functional coverage obtained. High assertion density only quantifies a value of coverage of the functionality specified, not directly relating to the actual functionality being verified. Moreover, a thing that is often overlooked when using assertions for FV is that writing correct and complete properties is the hardest part of obtaining a proof of correctness (or a constructive counterproof). Although this observation is especially pronounced in formal verification it also applies to assertion simulation.

With an assertion density reaching a couple of hundreds (which is plausible for a complex design) there is a likelihood of having requirements that do not fulfill their purpose – to give additional confidence of the designs correctness. Assertions might even be misleading as they may give false confidence, and draw attention from the problem at hand - to assure that the monitored functionality is behaving correctly, or not incorrectly. Misleading assertions can be due to a multitude of reasons: incorrect and incomplete requirements, incorrectly specified functionality, erroneous properties and incorrect assumptions on the design intent just to give a few examples.

The mere use of assertions for functional design verification does thereby not automatically provide convincing evidence that the required functionality is fulfilled without being validated against the functional design requirements and without being associated and analyzed within its context of the overall functionality.

Managing Complexity

With gate counts in designs growing steadily, incorporating larger input/output spaces and internal state holding elements *Programmable Logic Designs* (PLDs) are getting harder to test exhaustively within a foreseeable amount of time, which is what defines a complex design[2].

Even if the civil avionics domain is known to be cautious and mostly relies on proven concepts and hardly ever strive for making innovative designs steps that would jeopardize the overall safety objectives it still has to handle this increasing complexity and what many refer to as the increasing design-verification gap. The avionics domain therefore have a progressive view on verification and embrace new techniques, but at the same time use careful planning in assuring that the sought verification goals are met.

Discussions held within the verification community (not necessarily within the avionics domain) are suggesting that the solution to the increasing complexity of designs may even lie in the other end - in making better designs. To spend verification efforts where it helps designers create better designs from the start instead of hunting down the mistakes they make. This means for instance to focus on requirements capture and validation, maintaining good code quality by the use of coding rules and guidelines with a possible set of automatic code checks to see to that these coding rules are followed, maintaining good design documentation that captures the current design functionality during development to enable supportive verification, use of assertions to give instant feed-back to engineering to see that the design intent is followed and to perform regression testing when new requirements or a new design revision has been issued. Requirements capture/validation, design and verification processes can in other words be made in close relationship with each other. Synergy should be strived for while still maintaining implementation independent validation and verification.

Only maintaining a good quality of design is however not sufficient for safety-critical (DAL A and B) avionics hardware designs as certification compliance to DO-254 needs to approve certain

---

[2] *"A hardware item is identified as simple only if a comprehensive combination of deterministic tests and analyses appropriate to the design assurance level can ensure correct functional performance under all foreseeable operating conditions with no anomalous behavior. When an item cannot be classified as simple, it should be classified as complex."* DO-254 [1]

verification coverage and full traceability of results. For DAL A and B designs ideally all permutations and combinations of input/outputs, internal state combinations and functions should be tested. This means that as complex designs per definition cannot be tested exhaustively under a foreseeable amount of time additional design assurance strategies are needed.

Advanced Verification Strategies

DO-254 proposes among other solutions a number of advanced verification strategies, namely *Safety Specific Analysis* (SSA) that limits the tests performed to a subset of the input/out space, *Elemental Analysis* (EA) where the functionality is broken down into more manageable sub parts, and *Formal Methods* (FM) where formal proof calculus is used to automate the verification to a higher degree.

Apart from these verification methods there are many other means of verification available on the market to create advanced test cases and raise the abstraction level of verification. As example using simulation with assertions (that is the basis of the OLMM), using high-level coding languages and base libraries to create test benches, and constrained random stimulation just to name a few options. As mentioned above, especially the use of formal assertions to monitor design behavior during simulation has in recent years become more common in verification.

It needs to be pointed out that at the same time as you would like to raise the abstraction level of verification when creating test benches and when making test cases through the use of more expressive languages, hierarchical test architectures and by the use of more automated processes the focal point still has to be on the low-level design function being verified. Tests still have to be built bottom-up with the end functionality in mind, i.e. with signals behaving as intended, in accordance with the low-level hardware requirements.

In addition to the above verification methods there are a variety of coverage measures that can provide evidence that diverse aspects of the design are triggered during verification. Examples of such are code-coverage (line, conditional, expression, block), finite state-machine (transition, state), transaction-based and functional coverage metrics. All these metrics can be used to reason about the completeness of simulation based verification approaches. Although, when combined with FV where the results gained are either proofs/counterproofs of properties, or partial proofs of properties there is a mismatch in how to weigh these metrics against each other. The solution to this dilemma might be to look at the problem from the other side (as was previous suggested for the design-verification gap), to use a robust functional verification strategy and planning that assures that the coverage measures obtained are sufficient to demonstrate the designs correctness.

Functional Requirements Capture

Regardless if FV or assertion simulation is applied to the design under test the functional requirements on the design have to be formalized using a restrictive, unambiguous syntax. To support these activities the requirements as well as design documentation have to be described in detail to enable the verification team to work independently in creating formal properties.

As functional requirements are captured using a natural language such as English that do not obey to the strict rules as formal languages (PSL and SVA[3] for example) requirements easily leaves out information for the validation and verification engineer's interpretation. Because of this, translation between these two languages is needed, which is a potential source of errors and misinterpretations. By using formalized wording and a set of rules to capture functional requirements and their conditions might however make this translation easier by eliminating the most common mistakes.

It is however hard, or even impossible to write fully covering functional requirements for a complex design as it means describing the whole design functionality deterministically. This is not even possible using a formal language within a reasonable amount of time, and using a natural language this would mean a vast amount of requirements that would take ages to validate and verify. We thereby have to assure that the validation

[3] SystemVerilog Assertions [8]

made on the existing set of requirements assures that the complete design intent is captured.

Formal Specification

Formal properties are built-up by discrete sequences of events called *Sequential Extended Regular Expressions* (SEREs) that may be declared within variables to hierarchically create large functional expressions. Together with temporal operators these SEREs describe the events that are asserted on the design during verification.

Properties may also be constructed using a left-hand side (LHS) argument defining the preconditions under which the right-hand side (RHS) argument must hold. The LHS property arguments may also be used for functional coverage measures and constraining design behavior during FV. Both the LHS and RHS of the property argument are either constructed using the SEREs described above or with logical expressions.

When writing formal properties there are some things to consider. First of all the intended verification strategy has to be kept in mind as the way in which the properties can be written varies depending on it. If formal verification is intended, then the properties have to be made simple enough for the proving engine to be able to reach a conclusive result. On the other hand, if simulation is the verification strategy of choice, then the properties have to be limited to a subset of the formal language used to be able to be asserted by the simulation tool. It should at the same time be pointed out that the properties written for simulation may be re-used for FV, given that the tools used supports the same language constructs used (workarounds possible though) and that the property is simple enough (which is not always the case). You also have to bear in mind that different kinds of properties are suitable for different kinds of verification strategies. As example, a property checking that an event ‘x’ shall never occur can only give a proof of correctness either when exhaustively simulated (which is not possible with a complex design) or proven formally.

As there are differences between the properties written for the purpose of being simulated, and those that are to be formally proven it is suggested to keep these separate to each other. It is also suggested to write properties hierarchically by reuse of defined sequences of events may also eases the complexity as it allows the proofs to be calculated on the lowest possible functional block level. To be able to build properties hierarchical the SEREs can be declared as variables and combined into more complex sequences of events. In doing so the properties written also become more readable and more easily debugged as events are directly coupled with its respective function. For resuse of code and to make further use of these declared functions it is also suggested to parameterize these SEREs so that different functions can take use of these sequences. These sequences can thereafter be made available for global use throughout the design in reusable libraries.

As mentioned above, SEREs may be needed to constrain design behavior for formal proof-calculation. This way the input-space of the functional design block being formally verified is limited, which reduces the complexity of the proof. Intuitively, properties defined for functional blocks on the border of the PLD interfacing external inputs are often needed to be constrained so that the formal tool used can deliver constructive results. If the properties applied in FV are not constrained enough the proofs or counterproofs obtained from the formal tool are counterproductive and does not give additional value to verification. On the other hand, if the properties are over-constrained this means that important implementation aspects may be overlooked and verification engineers mislead to believe that complete proofs of correctness have been achieved. Proofs calculated under these false assumptions of course become of lessened value and may even have severe consequences. Constraining design functionality is therefore an important aspect of FV that has to be planned carefully and incorporated into review process. Another aspect of constraining design functionality is that to be able to formalize requirements the conditions for these to hold are needed to be defined in the requirements so that its completeness can be validated. The constrained properties are also suggested to be kept separate to unconstrained properties so that the proofs calculated are made given the right preconditions, but also to ease the workload of proof calculation, debugging and analyzing results as well as for configuration management reasons.

It is thereby imminent that an equally, or more, strict procedure in writing formal properties than is used for HDL coding is needed. This involves the use of clear and precise coding rules that are linked to the way requirements themselves are captured.

Functional Modeling

To support the functional requirements capture/validation and the specification of formal properties functional modeling in the *Unified Modeling Language* (UML) can be used to graphically visualize the design intent. As described in the UML reference manual [9], models can be used for (among other purposes):

- *"To capture and to precisely state requirements and domain knowledge so that all stakeholders may understand and agree on them."*
- *"To think about the design of the system."*
- *"To master complex systems."*

Bridging the gap of what is required of the design (non-formally and formally), and what the actual functionality is within the same environment enables a better visibility of how all the process activities correlate and it thereby assists in assuring that all vital functions are covered, that the formalized behavior of the design is correctly interpreted and that assumptions made on the design behavior is kept to a minimum. This is especially important considering that with independent verification (as is needed for level A and B designs) comes the disadvantage of insight into the implemented functionality. Validation and verification engineers might therefore be unaware of what they are missing due to lack of information, documentation and detailed knowledge of the design implementation.

Functional Test Plan

By making functional test trees of the design that incorporate functional requirements the correctness of requirements, design intent, formal properties and verification strategies can all be reviewed in relation to each other, which contribute to the quality of all processes during development.

Modern modeling tools today facilitate plug-ins to enable linking to the common requirements capture environment used within the avionics market, i.e. DOORS. With such features the functional requirements captured can be included within the model and updated automatically when new baselines of requirements are released. This of course supports the iterative requirements capture process in relation to the design and validation processes and is essential for configuration management of hardware design items.

Functional test trees make the planning of verification strategies explicit and facilitate the assigning of appropriate verification strategies to individual functionalities of the design. As is common procedure for complex HDL designs to break-up the functionality into functional blocks, so can the verification strategy planning. This verification scheme is of course something that can be revised during the verification process as it is as much gain for planning, as it is for documentation of the overall verification results.

Via building the functional test trees hierarchical with a block-level on the very top representing the main functions of the design the functionality can be broken-up hierarchically into more manageable subparts (compare with Elemental Analysis verification proposed by DO-254). At the top-level view the verification strategy planning can be outlined with assigning suitable verification strategies to the individual functional blocks.

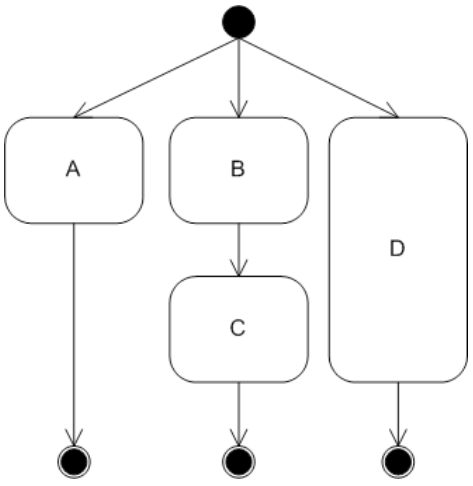

Figure 1 Schematic Block-level functional verification plan.

In Figure 1 above a generalized schematic example of a functional verification plan is outlined. The block level functions (A-D) within the model all represent different functions within the design. They might not be totally independent as the figure might be leading you to think, but from a verification planning point of view it is possible to treat these functions differently. For example, function A above might a simple function if treated on its own (not considering the systems it is residing in). Simulation methods might here be sufficient to exhaustively test this function. Function B and C on the other hand, have to be verified together and as C is depending on input coming from B. It is thereby possible that the functions within C are hard to stimulate from external inputs. With guidance of functional coverage it might therefore be needed to target those events that are specifically hard to reach with FV. Furthermore, let's say that function D is a complex function that cannot be exhaustively verified using only a traditional test bench. For this function a multitude of verification strategies might be needed to achieve coverage goals.

As mentioned above there is a variety of verification approaches available on the market and choosing an appropriate strategy is not always the easiest of tasks. The most suitable choice for verification depends, among other things, on the type of function being verified, the complexity of the design, the possibility to stimulate the particular function from design inputs, and the way the function has been implemented (which is unknown to an independent verification engineer).

As example, use of FV might be needed to reach the most hard to reach corner-cases, or to give additional confidence to aspects of the design of particular safety-critical impact. By using functional test trees for verification planning these verification "hot-spots" can be identified and targeted. The use of FV itself also involves many other alternatives: the verification engine to be used to calculate the formal proof, the constraints under which the proofs shall be made, and the way the proofs have been obtained, i.e. automatically calculated by the tool or manually guided. The latter of these, manually guided formal proofs also require detailed design knowledge, which is not always available to an independent verification engineer. In this case a functional test tree aids the verification engineer in judging the design's correctness.

Note that all the verification strategies applied to the design have to be placed under configuration management to assure that verification results obtained are traceable and configuration/version controlled.

Functional Test Trees

The block-level functional test plan can then be broken-down into more detailed functional test trees that give more detailed knowledge of the individual functions. The level of detail needed for these trees is something that has to be analyzed from the start of the verification planning to give sufficient information about the elements of the design to support the verification strategies used for the particular function.

In Figure 2 below a schematic example of a UML functional test tree is given. The example contains four states (A, B, C and D) with respective events to trigger the state transitions between these states. From the starting state, as depicted in the UML constraint (1 in Figure 2), from state A, when 'a' occurs with three consecutive 'b':s then the requirements in state D (2 in Figure 2) shall hold.

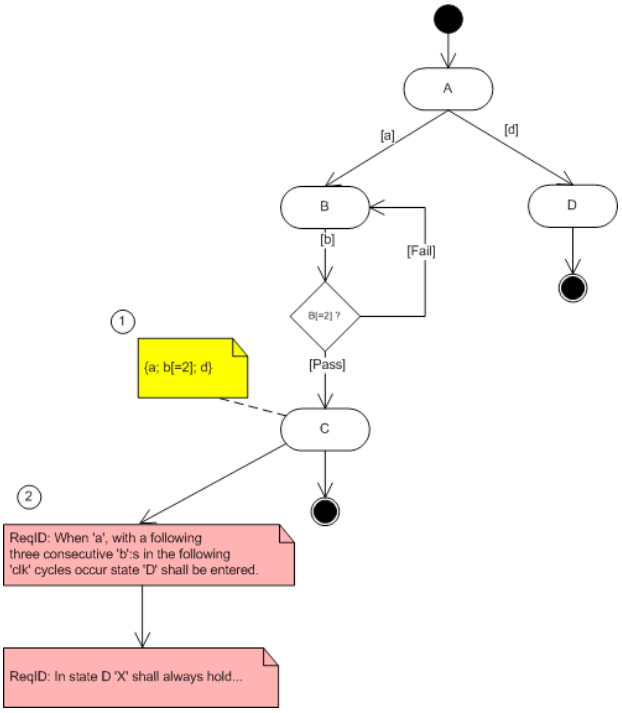

Figure 2 Schematic view of functional test tree.

As seen in the example above, requirements can be placed into their appropriate context when added in functional trees. The requirements can also be imported automatically from requirements baselines, and the trees might thereby be kept updated through the validation process. Which aids the requirements capture process, and assist in validating the requirements correctness and completeness for the intended functionality.

By specifying the design functionality in a tree structure the constraints, i.e. preconditions, under which the requirements must hold can be easily recognized. When constraining design input for formal proofs and random stimulus generation the inputs are limited to a subset of the complete input space (compare with Safety-Specific Analysis proposed by DO-254).

Note that formal properties can be applied on several levels of the design. If sub trees are used to describe the functional design and part it into manageable parts, possibly for verification complexity reasons, the property structure may follow the same structure.

In association to the functional tree structure formal properties coverage measures and preconditions (LHS) follow the branches of the tree and measure the functional coverage in comparison with the design intent. Thereby, the functional trees can be used for functional coverage planning and direct assertion coverage efforts to bring higher value to the quantitative assertion density measure. In this manner a clearer picture of what functionality has, and has not, been captured in properties is attained.

Conclusions

We have in this paper proposed an outline to a verification planning process to be able to draw advantage of the various functional verification strategies that are out on the market. With this structured verification planning process we have more possibilities to analyze the design intent, and validate functional requirements in their functional context as well as to assign appropriate verification strategies to functions where they are best suited.

Without careful functional verification planning the best we can accomplish using formal properties is to gain a quantitative measure of assertion density, and with FV tools try to analyze proven and counter-proven properties within its functional context. Writing these formal functional properties and constraining the verification effort is the most difficult aspect of effectively using FV, which requisites a careful planning and validation process.

By using a functional verification planning like the one proposed here it is also possible to weigh and review the functional coverage obtained from different verification methods altogether. Thereby, the coverage measures from FV may be reviewed in relationship to the coverage metrics acquired from simulation.

References

[1] Radio Technical Commission for Aeronautics, RTCA Special Committee 180, April 2000, *“Design assurance guidance for airborne electronic hardware,”* RTCA, Inc., Washington, DC, USA.

[2] Karlsson K. and Forsberg H., Oct. 2005, ”*Emerging verification methods for complex hardware in avionics*”, 24th Digital Avionics Systems Conference (DASC).

[3] Karlsson K. and Forsberg H., Oct. 2006, *"Increasing Confidence of Complex Hardware in Safety-Critical Avionics Using Formal Methods"*, MAPLD 2006[4].

[4] *"IEEE Standard for Property Specification Language (PSL)"*, 2005, IEEE Computer Society Std. 1850.

[5] Dewey T., *"Demystifying DO-254"*, Mentor Graphics Corporation, 2008.

[6] Lange M., Dewey T., *"Achieving Quality and Traceability in FPGA/ASIC Flows for DO-254 Aviation Projects"*, IEEEAC Paper #1412, Version 1, Oct. 2007.

[7] Keithan J. P., Landoll D., Marriott P., Logan B., *"The Use of Advanced Verification Methods to Address DO-254 Design Assurance"*, IEEEAC Paper #1304, Version 2, Sept. 2007.

[8] *"Standard for SystemVerilog - Unified Hardware Design, Specification, and Verification Language"*, IEC 62530 Ed. 1 (2007-11) (IEEE Std 1800-2005).

[9] *"The Unified Modeling Language Reference Manual"*, Rumbaugh J., Jacobson I., Booch G., Addison Wesley Longman, Inc., 1999.

E-mail Addresses

kristoffer.ko.karlsson@gmail.com

hakan.forsberg@saabgroup.com

Conference Identification

Military and Aerospace Programmable Logic Devices (MAPLD) Conference

September 15, 2008

[4] Unpublished due to MAPLD conference break, notify authors of this paper to receive referred document.