



ACCELERATING DO-254 VERIFICATION



Blue Pearl Software

ACCELERATING DO-254 VERIFICATION

INTRODUCTION

Automated electronic control systems or avionics allow lighter, more efficient aircraft to operate more effectively in the current and future airspace. Today's modern aircraft use an increasing number of microcomputers, microprocessors, and dedicated hardware to process growing amounts of data needed to control and monitor the status of the flight and related systems. In April of 2000, a joint RTCA Special Committee and EUROCAE working group released the Design Assurance Guidance for Airborne Electronic Hardware. In 2005 the FAA formally recognized RTCA DO-254 as a means of compliance for the design of complex electronic hardware in airborne systems with a goal to establish clear objectives to ensure development of safe and robust avionics equipment.

The guidance is intended to be applied across line replaceable units, circuit card assemblies and integrated circuits, with the goal of ensuring that custom components, such as an ASIC or FPGA, meet intended functional and safety standards for airborne applications. The DO-254 specification utilizes a requirements-based design and verification approach. This means that the entire hardware project revolves around a formal set of high-level requirements. Prior to writing RTL, each of these requirements must be captured, given a unique reference name, and reviewed for a variety of criteria including understandability, testability and verifiability.

The standard defines the hardware design processes and breaks it into these phases:

- Requirements Capture
- Conceptual Design
- Detailed Design
- Implementation
- Verification
- Transfer to production

The more complex an ASIC or FPGA becomes, the more complex and sophisticated the tools supporting its development and verification must be. Verification is intended to check that a system, or a portion thereof, meets a set of design specifications. Verification of a hardware design can find errors at a variety of levels. Errors can be identified in the hardware requirements, translation of the design into register transfer language (RTL), or implementation of the RTL in hardware. To be able to certify the functionality of a system with design assurance level A or B, there are several different analyses that must be performed for complex systems. All of these analyses are iterative and must be reevaluated whenever a design change is made.

TYPICAL HARDWARE DESIGN FLOW

The typical hardware design and verification flow is illustrated below, with verification today taking 50% to 70% of the overall development time. By applying static, dynamic and formal verification practices proven to reduce develop time, design iterations are reduced and design quality is increased.

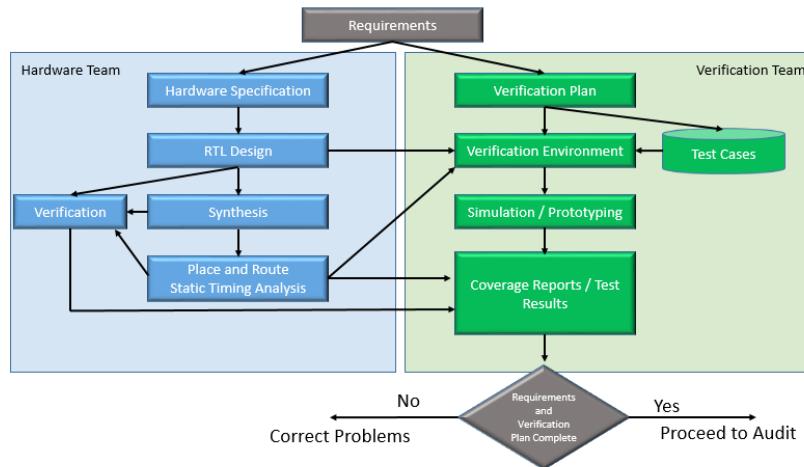


Figure 1 – Typical DO-254 Hardware Design Flow

RTL VERIFICATION

RTL checking is an integral part of ASIC/FPGA design team workflows. DO-254 suggests that coding standards should be defined and reviewed. However, the policy does not clarify what these standards are and certification authorities usually do not have a good set of industry standard rules to recommend. Design teams or individual designers are then left to define a set of rules for RTL that ensures code quality. For example, when manually inspecting code, design teams may check for common coding errors, such as incomplete FSMs, latch inferencing, and incomplete sensitivity lists (which can cause mismatches between simulation and synthesis results). However, manual checking can be error prone and result in issues being found late in the design flow.

RTL coding practices can also affect timing closure. Long If-Then-Else structures or modules with unregistered ports, for example, will pass through simulation and synthesis without errors only to produce long paths in the design. It can take days to identify the root cause looking through static timing results.

Modern ASIC or FPGA designs also usually contain many IP cores developed by multiple teams either internal or external. Establishing and applying unified design rules for coding style is key for accurate simulation and synthesis, and it improves the readability and maintainability of RTL. The quality of the IP for complex systems is critical during integration. Leveraging a good lint tool can ensure correct-by-construction design practices

Recommend RTL checks include:

- Avoiding FSM state unreachability, terminal states, and coding issues
- Avoiding differences between simulation and synthesis semantics
- Avoiding operations with expensive implementation costs
- Following naming and RTL coding conventions
- Enforcing RTL modeling clarity and reducing complexity
- Enforcing checks for clock bundles (clocks, enables, resets) and control signals
- Enabling testability and traceability of the code
- Avoid long ITE chains
- Confirm all IP ports are registered

CROSS DOMAIN CLOCKING ISSUES

Simulation requires the generation of appropriate test vectors and is an accepted traditional method for functional verification during the design creation phase, however it may not catch all design issues. Verification of the hardware using simulation may consist of both directed test vectors and randomly generated vectors. This method is entirely adequate to verify that the design specified in RTL performs the intended function within the limits of simulation. However, simulation alone can miss critical issues that can cause metastability issues late in the system integration or, worse, after the system has been deployed. While it is possible to find a CDC and then create a test vector, finding CDC issues using simulation is virtually impossible.

Ideally, a design will have only a single master clock. Unfortunately, modern designs commonly require several independent clocks used within a single system. When signals move from one clock domain to another, special circuits and analyses are required.

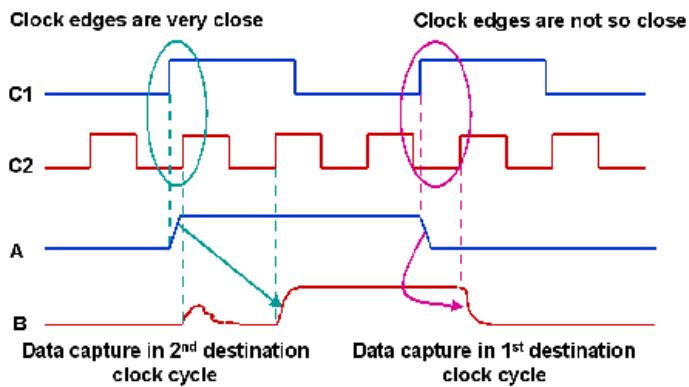


Figure 2 – Clock Domain Crossing without Synchronizer

Modern Clock Domain Crossing tools point out where synchronizers are required to allow signals to cross clock domain boundaries. Correct operation of circuits crossing clock domain boundaries cannot

be guaranteed by simulation because the timing between different clock domains can vary arbitrarily. This would require an infinite number of simulations to verify.

TYPICAL HARDWARE DESIGN FLOW LEVERAGING VISUAL VERIFICATION SUITE

The key to design assurance using DO-254 is identifying all possible design errors. Before a design error can be identified, a test case that produces the error must be generated, and then the error must manifest itself in a way that the test system can detect. Blue Pearl's Visual Verification Suite's (VVS) Analyze RTL incorporates a suite of technologies into a single RTL analysis environment to help developers find bugs earlier in the design processes. Super-lint tools are combined with the power of formal verification to provide a single, high capacity design checking environment that identifies poor coding styles, improper clocks, simulation and synthesis problems, poor testability and other source code issues. FSM analysis automatically extracts and analyzes finite state machines for dead or terminal states and provides a visual representation. X-propagation analysis detects unknown states, often introduced into designs to implement soft resets or to implement power management schemes that are masked during RTL simulation.

The VVS Advanced Clock Environment (ACE) and CDC Analysis provide graphical representations summarizing data paths between clocks, and can make recommendations for grouping of clocks into clock domains. With ACE, designers can identify clocks to better understand how they interact with synchronizers in the design. This allows users to quickly identify missing synchronizers or improper clock domain groupings. The VVS CDC Analysis incorporates over a decade of experience to find errors other tools may fail to identify.

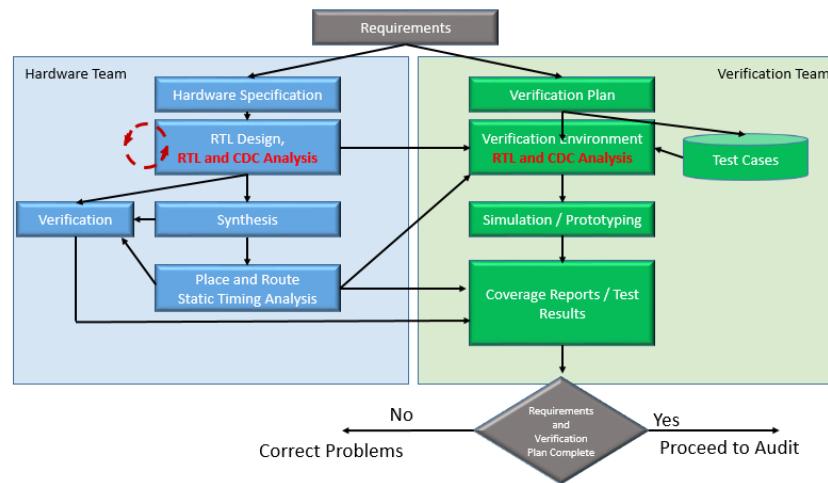


Figure 3 –DO-254 Hardware Design Flow Leveraging VVS

INDUSTRY'S ONLY STATIC VERIFICATION TOOL OPTIMIZED FOR FPGA AND ASIC

The Visual Verification Suite offers the only Linting and CDC verification environment optimized for both ASIC and the unique requirements of FPGA design. Specific for FPGA are checks to analyze for routing congestion, reset configurations and even estimate critical timing paths prior to synthesis. Grey Cell modeling supports the analysis of FPGA vendor and 3rd party provided protected IP cores with asynchronous clock domains. The Visual Verification Suite supports Xilinx Vivado™ Design Suite with built in UltraFast™ Design Methodology design rules, Microsemi Libero Design Suite and Intel Quartus® Prime Design Software and is the only static verification environment that runs on Windows – the preferred FPGA environment – as well as Linux operating systems.

For more information about Grey Cell Modeling see [RTL Analysis for Complex FPGA designs using a Grey Cell Methodology to Improve QoR](#)

QUALIFYING BLUE PEARL SOFTWARE FOR DO-254

Tool vendors do not qualify their own tools under DO-254. If you intend to use the Visual Verification Suite on a DO-254 project, here are some suggestions for getting through audits.

- 1) Do not qualify. If your project requirements do not call out the need for VVS, then it does not need to be identified and described in the “Plan for Hardware Aspects of Certification” (PHAC) or other DO-254 documents. Unless you have a specific requirement for linting and/or verification of CDCs, you can just leverage VVS without it becoming part of the DO-254 review process.
- 2) If there is a specific requirement from your customer (or even your DER) that says you must use linting or CDC tools then you must choose a method of tool assessment. The simplest one by far is Independent Output Assessment. In this case, you would specify something similar to the following:

“The results of the VVS are checked independently by thorough testing with 1000’s of independent tests using both false positive and false negative testing. We have/will be conducting tests in hardware under real system clocking frequencies and checked that all the conditions that were analyzed operate as expected.”

For more information on Blue Pearl’s Software’s reliability testing see: [How Can We Build More Reliable EDA Software Whitepaper](#)

BLUE PEARL SOFTWARE

Blue Pearl Software, Inc. is a provider of design automation software for ASIC, FPGA and IP RTL verification. Its Analyze RTL™ linting and debug, Clock Domain Crossing analysis and Synopsys Design Constraints (SDC) generation solutions are proven to improve quality-of-results (QoR), reduce risk and decrease development time. The Visual Verification Environment complements RTL simulation solutions provided by EDA and FPGA vendors by ensuring code and SDC quality along with clocking integrity. Engineered to maximize RTL find/fix rates, the Visual Verification Suite uniquely provides easy setup, consistent results, Management Dashboard for complete push-button analytics, and runs on both Linux and Windows.

VISUAL VERIFICATION SUITE DO-254 PACKAGE

PACKAGE: DO-254

Description: This package includes checks that aid in compliance with the DO-254 military standard.

Checks:

- GRST (Gated reset)
- HCCC (Do not hard-code constants)
- REGO (Register all module outputs)
- UNREACHABLE_STATE (Report Unreachable states)
- MCD (No case default)
- MDA (Missing case default assignment)
- CSL (Complete sensitivity lists)
- MEB (Missing else block)
- LEC (Little endian checks)
- CCLP (Infinite and 0-count loops)
- SLCC (Separate lines for commands)
- INT_TRI (Internal tristate objects)
- COMMENT_END_STMNTS (Comment end statements)
- MCA (Missing case item assignment)
- MIA (Missing if assignment)
- COMMENT_NET_DEC (Comment net declarations)
- NS_NAME (Next state names)
- COMMENT_PORT_DEC (Comment port declarations)
- RST (All flip-flops resettable)
- RESET_POLARITY (Check reset polarity consistency)
- FOREIGN_LANGUAGE_KWD (Report usage of Foreign Language keywords as identifiers)
- CLPR (No combinational loops)
- MULT_MODS (Multiple modules in the same file)
- FILE_HEADER (Use a file header)
- CONSTRUCT_HEADER (Use a construct header)
- TERM_STATE (Report Terminal states)
- MEA (Missing else assignment)
- NO_TABS (Do not use tabs)
- IODECL (Mixed input/output/inout port declarations)
- POSEDGE (Use positive edge clocks)
- IGRST (Internally generated resets from different modules)
- IDDC (Isolate 'define' directives)
- VHDL_CHECKS (Report VHDL naming rule violations)