DOT/FAA/TC-19/46

Federal Aviation Administration
William J. Hughes Technical Center
Aviation Research Division
Atlantic City International Airport
New Jersey 08405

Tool Qualification Issues for Airborne Electronic Hardware Programmable Logic Devices

February 2020

Final Report

U.S. Department of Transportation
Federal Aviation Administration

NOTICE

Technical Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
DOT/FAA/TC-19/46		

4. Title and Subtitle	5. Report Date
Tool Qualification Issues for Airborne Electronic Hardware Programmable Logic Devices	February 2020
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
Mark Fischler, Mark Edel, Bill Haynes, Laurence H. Mutuel	Deliverable 11

9. Performing Organization Name and Address	10. Work Unit No. (TRAIS)
Thales Air Traffic Management 10950 El Monte Street, Suite 110 Overland Park, KS 66211	11. Contract or Grant No. DTFACT-13-D-0008

12. Sponsoring Agency Name and Address	13. Type of Report and Period Covered
FAA National Headquarters 950 L'Enfant Plaza N SW Washington, DC 20024	Final Report
	14. Sponsoring Agency Code AIR-130

15. Supplementary Notes

The FAA William J. Hughes Technical Center Aviation Research Division Technical Monitor was Manny Rios.

16. Abstract

Programmable Logic Devices (PLDs) are commonly used custom logic components in aerospace applications. Developers commonly use tool suites that facilitate device design and provide for automated synthesis, simulation, and testing of device configurations. These tools are essential for reliable development of a device that fulfills a complex set of requirements. However, it may be difficult or impossible to demonstrate tool qualification, per DO-254, for these tools.

Demonstrating safety assurance for PLDs should combine hardware and software aspects, incorporating techniques akin to DO-178C software assurance. Even if DO-254 were amended to recognize DO-330 as a guideline, tool qualification would present difficulties because vendors providing tool design artifacts are not driven by the small aerospace market and the preponderance of PLD development tool service experience arises in non-aviation contexts.

This report relies on detailed analyses of design flows for specific field programmable gate arrays, and inputs from design engineers based on numerous programmable logic development projects, to assess current practices and identify particular risks of defect injection and tool reliability concerns. Current practice commonly skirts the issue of qualification of the netlist synthesis and place and route tools by providing independent assessment of their outputs and successful testing of the resulting configured device. Existing tools, though not necessarily qualified in the DO-254 sense, have proven reliable in many respects.

Appreciable levels of risk of defects occur because of a combination of suboptimal tool interface and imperfect attention to detail on the part of the developer, incomplete specification of timing constraints, ambiguous specification of states, and reliance on flawed prepackaged logic blocks. There are opportunities to improve safety and reliability by dealing with these risks, and by making the assurance of synthesis and verification tools a more uniform process. Twenty-five specific recommendations are presented to achieve these goals.

17. Key Words	18. Distribution Statement
DO-254, DO-330, design assurance, AEH, tool qualification, FPGA, complex PLD, tool assessment, service history, test vector, verification, simulation, multi-DAL components.	This document is available to the U.S. public through the National Technical Information Service (NTIS), Springfield, Virginia 22161.

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	100	

ACKNOWLEDGEMENTS

Acknowledgement is due Matt Klueppel at Thales Visionix, Jean-Marc Gagne and Elias Rederick at Thales Group Aviation (Montreal), Abhyudaya Reddy Eluganti at Thales ATM, and Cyril Marchland at Thales Avionics (France) for experience-based information concerning development of field programmable gate arrays (FPGAs) for airborne applications. Acknowledgement is also due Jeff Plane for insights about treatment of analogous safety assurance issues for military aircraft. Particular acknowledgement is due Joshua Einstein-Curtis, Alan Prosser, Kurt Biery, and numerous other FPGA engineers for conveying a broad spectrum of details and experience concerning risks of defects introduced in the course of FPGA design.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS AND ABBREVIATIONS

ARINC	Aeronautical Radio, Incorporated
ARP	Aerospace Recommended Practices
AWA	Airworthiness Authority
CAST	Certification Authorities Software Team
CPU	Central Processing Unit
CVT	Compliance Verification Toolset
DAL	Design assurance level
DER	Designated engineering representative
DMA	Direct Memory Access
EDIF	Electronic design interchange format
FEC	Focused expression coverage
FPGA	Field programmable gate array
GPU	Graphics processing unit
HDL	Hardware Description Language
HLS	High-level synthesis
ICU	Interface control unit
IDE	Interactive development environment
IDP	Interface Display Processor
IP	Intellectual property
ISE	Integrated System Environment
LCOS	Liquid crystal on silicon
LVDS	Low-voltage differential signaling
MCDC	Modified condition decision coverage
OFP	Operational flight program
PCI	Peripheral component interconnect
PD-175	Personal display 175
PLD	Programmable logic device
P&R	Placement and routing
RTL	Register transfer language
SMARC	Smart Mobility Architecture
SMPTE	Society of Motion Picture & Television Engineers
SOW	Statement of work
UML	Unified modeling language
VHDL	VHSIC Hardware Description Language
WDC	Wideview Display Controller

EXECUTIVE SUMMARY

Programmable logic devices (PLDs) are commonly used custom logic components in aerospace applications. Developers commonly use tool suites that facilitate device design and provide for automated synthesis, simulation, and testing of device configurations. These tools are essential for reliable development of a device that fulfills a complex set of requirements. Defects in synthesis tools may lead to injection of defects, and defects in simulation or testing tools may lead to failure to detect improper behavior. However, it is difficult or impossible to demonstrate tool qualification, per DO-254, for every tool. Current practice often relies on claims of independent assessment of tool outputs, or on assurance techniques that are appropriate for simpler hardware devices. This leads to a spectrum of assurance activities and a set of de facto guidelines that are not uniform across similar development projects.

Demonstrating safety assurance for PLDs, including field-programmable gate arrays (FPGAs), should combine hardware and software aspects, incorporating techniques akin to DO-178C software assurance. If the device configuration were considered to be software development, then DO-330 would apply as guidelines for qualifying design tools. Even if DO-254 were amended to recognize DO-330 as applying, qualification of these design tools would present difficulties because the preponderance of PLD development tool service experience arises in non-aviation contexts and may not be deemed relevant for FAA safety-assurance purposes. Process-based qualification would also be difficult because vendors who make tool design artifacts available are not driven by the small aerospace market.

This report relies on detailed analyses of design flows for specific FPGAs and inputs from design engineers based on numerous programmable logic development projects to assess current practices and identify particular risks of defect injection and tool reliability concerns. Safety assurance is achieved if it is unlikely that defects have been introduced, and that verification activities cover all requirements and would likely expose any existing defect. Proper tool performance is relevant for both of these layers of assurance.

Current practice commonly skirts the issue of qualification of the netlist synthesis and place and route tools by providing independent assessment of their outputs, in the form of a combination of manual review of outputs and successful testing of the resulting configured device. This combination is generally deemed acceptable by designated engineering representatives. However, that acceptability may partially be influenced by the realization that tool qualification is at this time impractical.

Existing tools, though not necessarily qualified per DO-254, have proven reliable in many respects. Although one can identify quite a few flaws that were injected, at least temporarily, during the design process, none of these were found to be due to improper performance of a properly used design tool. A similar statement holds for flaws that validation steps failed to expose. However, these observations for general usage of the design tools are not as valuable as project-specific qualifications of each tool would be.

Appreciable levels of risk of defects occur in four areas. Some problems occur because of a combination of suboptimal tool interface and imperfect attention to detail on the part of the developer. There are issues associated with incomplete specification of timing constraints.

Multiple clock domains present possibilities of ambiguous specification of states and of incomplete reset activities. Finally, reliance on flawed prepackaged logic blocks, such as state machines or bus and memory controllers, can lead to injection of defects.

There are opportunities to improve safety and reliability by dealing with these risks, and by making the assurance of synthesis and verification tools a more uniform process. One general recommendation is that guidelines applicable to complex PLDs ought to recognize that the development process is similar to that of software. Guidelines should specify parts of DO-178C and DO-330 that are applicable to development of such devices. This should be specified regardless whether a revision of DO-254 may cover similar issues.

Prospects for qualification of mature netlist synthesis tools are reasonable, but place and route tools must undergo frequent revisions as chips evolve. It may be wise to have some "register" of combinations of chip versions and tool releases that have been sufficiently tested by experience to be considered to have superior reliability. It is felt that qualification of detailed timing simulators for evolving devices is not feasible. Results of timing simulations should be accepted as verification evidence only if cross-verified by actual hardware timings.

This report makes 25 specific recommendations. Recommended steps to reduce the impact of identified risks and to facilitate tool qualification or achieve the equivalent level of safety in that area fall into four areas:

1. Improved design and verification activities include a required review of reset behavior and clock domains, strengthened validation of the gate layout by examining timings in critical paths, qualification of tools used for test vector automation, and cross-verification of timings and simulations for high Design Assurance Levels (DALs).
2. Improvements related to treating PLD development as software include the use of a subset of DO-178C assurance techniques, treating timing constraints as source code subject to review and configuration management, and providing evidence of reproducible device configuration from sources.
3. There are several recommendations for guidance aimed at tool vendors, including clarification of what development/testing artifacts can facilitate tool qualification, allowing for qualification of tool kernels, and clear requirements of active errata presentation and history/defects documentation to support qualification.
4. The remaining recommendations deal with activities that pose particularly high risks of defect injection or conceptual difficulties for practical safety assurance. The developer should treat "public" intellectual property blocks as advisory, taking control of the source code and performing review and validation activities. Even blocks obtained from tool vendors require tool qualification or independent assessment. Finally, there should be analytic verification of state machines and proof of separation for devices incorporating multiple design assurance levels.

Implementation of a selected subset of these recommendations may improve safety assurance of the development of PLD configurations, particularly in the areas affected by the use of development tools, without imposing undue burdens on the development process.

1. INTRODUCTION

1.1 PURPOSE

This evaluation effort has been undertaken in response to a statement of work (SOW): Programmable Logic Device (PLD) Tool Qualification, Delivery Order #7 under DFACT-13-D-00008 [1]. Per that statement, the purpose of this research is to determine the effectiveness of existing verification activities for typical design flows in detecting all classes of errors that may be inserted by an unqualified design tool.

This report uses analysis of detailed design flows for specific PLDs and provides research on existing practice in PLD component development with inputs from design engineers based on numerous PLD development projects. These analyses are used to identify error classes that could potentially be introduced by the use of unproven design tools. The information is used to assess risks of defect introduction and identify possibilities of errors that may escape detection using existing prescribed verification activities. In particular, the netlist synthesis and place and route activities, if flawed, may introduce errors. Although errors resulting in repeatable misbehavior would likely be exposed by verification activities, subtle violations of timing rules, for example, might not be detected.

Recommendations are presented for additional and improved verification activities to address these risks. Recommendations for clarification of tool assessment and qualification guidelines, to accurately reflect safety assurance needs in the context of current design practices, are also presented. These indicate ways to streamline the safety assurance for field-programmable gate arrays (FPGAs) with regard to the use of design tools.

Where the SOW refers to “verification activities” involving tools, this research considers a broad definition of those activities. The term “verification” would ordinarily refer to that phase of design in which the device creation is complete and the resulting device is thought to be correct; verification generates proof of that correctness for use in safety assessment, or identification of deficiencies that must be addressed. In the inclusive definition used for this research, tool-assisted synthesis and preliminary validation activities are also considered because the issues concerning possibilities of flawed tools injecting defects into the device (or allowing defects to remain undetected) are similar to the issues concerning flawed verification tools.

The analyses and recommendations in this report are based on research on existing practice in PLD component development, detailed analysis of the design and verification processes used to develop two key FPGA components, identification of potential errors, and assessments of the magnitudes of risks associated with the use of unqualified tools.

1.2 BACKGROUND

PLDs are commonly used custom logic components in aerospace applications. PLD developers commonly use tool suites that facilitate device design and provide for synthesis, simulation, and testing of PLD circuits. Developers’ attempts to explicitly perform the functions automated by these design, simulation, and testing tools almost always lead to unreliable PLD configurations. Therefore, these tools are essential for development of a PLD that fulfills a complex set of requirements. Among the activities that rely on design tools are:

- Synthesis—Hardware description language (HDL) code, which is closely tied to low-level device logic requirements, is translated to a list of connections among gates, flip-flops, and other basic electronic elements that provide a logic gate implementation of the HDL code. This list of electronic elements and connections is called a "netlist."
- Logic element placement and routing (P&R)—Mapping the netlist elements to the FPGA physical resources, placing the resources within the FPGA, and routing the connections. This P&R must obey timing, signal route, current density, and other constraints specific to the FPGA type used to implement the hardware device.
- Simulation—Software assessment of the results achieved when input signal sequences representing tests of requirements are processed by the synthesized and routed FPGA. This simulation is often used to create a back-annotated netlist file, with timing information that can be analyzed to verify proper performance.

Demonstrating safety assurance for PLDs combines hardware and software aspects. As is the case with any electronic device, there is a broad class of possible flaws involving improper circuit design and single random hardware failure events. There is also the possibility that environmental conditions prevent proper execution. For example, an FPGA that is warranted by the manufacturer to behave within specifications in some temperature range might encounter temperature extremes outside that range in aircraft applications. Such issues apply to the use of any hardware component, and the acceptable means of assessing risk factors for failures mentioned in DO-254 (e.g., section 2.3.2) are applicable to and necessary for complex PLDs. These issues, which are universal to any airborne electronic hardware, are not within the scope of this report.

For the software aspect of configuring FPGAs, however, considerations pertinent to software development are logically relevant. Assurance techniques akin to those accepted in AC 20-115C (e.g., DO-178C) may become applicable. Because of the essential role of sophisticated PLD design tools, there is an additional broad class of possible faults that could lead to incorrect performance of a PLD. Flaws in the translation of the high-level functionality specifications provided to the tool, into a correct logic net, can lead to execution of faulty functionality. Flaws in the P&R of logic elements can lead to incorrect or unreliable execution of the desired logic.

Although chapter 11 of DO-254 [2] provides guidance for tool assessment and qualification for design and verification tools, PLD development tools are rarely qualified in current practice [3]. In part, this is because there is no obvious appropriate path to qualification of these as hardware development tools. Developers often rely on claims of independent assessment by alternative means of verification of their output. Section 11.4.1 in DO-254 states: "Independent assessment of a design tool's output […] may be established by the verification activities performed on the item, such as component, netlist, or assembly. In this case, the integrity of [the] end item does not depend on the correctness of the design tool output alone. Independent assessment of a verification tool output may include a manual review […] or may include a comparison against the outputs of a separate tool capable of performing same verification activity as the tool being assessed."

However, reliance on independent assessment of a complex design tool's output may not be wholly satisfactory as a means of assuring that the design is error-free or that all potential errors that might be introduced by defects in the tool have been detected and remedied. Nor is universal reliance on this approach to verification necessarily optimal in terms of design efficiency. Improved verification activities, and possibly changes in the tool-qualification process so that the

qualification approach can be practical in appropriate situations, might help to reduce the chances of undetected design flaws and allow developers to focus their verification activities more optimally.

Because development of the configuration of PLDs such as FPGAs has much in common with software development, the considerations in DO-330 "Software Tool Qualification Considerations" [4] might be appropriate in many cases. However, DO-330 is linked to DO-178C [5] rather than DO-254; therefore, there is no firm justification for using the criteria of DO-330 to qualify these design and verification tools. There may also be a mismatch between the pace of evolution of FPGA design tools, which necessarily must be modified to accommodate each new variant of FPGA, and the viewpoint of DO-330, which may consider each modification as creating a tool needing its own qualification.

Assessment of a design tool's output based solely on efforts of the PLD development team in the context of the device creation project excludes the experience of others regarding the use of that tool, and does not exploit information that may be available to the tool provider and/or the general user community. A tool-qualification process presents an organized opportunity for incorporating such inputs and may augment the effectiveness of the process of assessing the tool's output.

In addition, sophisticated tools are used in major parts of the verification of any complex PLD via simulation and analysis of the detailed simulation results. DO-254 states that simulation results alone cannot be used for the purpose of certification credit without supporting evidence of their validity. Whereas agreement with the results of hardware testing can be considered supporting evidence, the kinds of study going into a tool-qualification process tend to expose a different set of potential tool flaws (or show that those flaws are not present) than would easily be revealed by hardware testing of the designed device. The value of such a review or comparison is acknowledged in a note to section 11.4.1 paragraph 3 of DO-254, though the main thrust of section 11.4 is tool qualification.

1.3 STRUCTURE OF THIS REPORT

This report provides information and develops recommendations relevant to FAA safety assurance of complex PLDs. Particular emphasis is placed on issues of design tool qualification and safety assurance of designs created with the help of unqualified tools.

Section 2 describes the activities performed in the course of designing the configuration for an FPGA. After providing a generic framework of synthesis, validation, and verification activities, a detailed analysis of the design flow for two specific FPGAs designed for airborne operation is presented. Observations based on the example design flows discussing types of errors observed, and tool-related concerns, are presented.

Section 3 classifies risks of defect injection and risks of failure to detect improper operation of the FPGA that developed. In section 4, a table of vulnerabilities, including risk levels, provides a basis for discussion of relevant safety assurance considerations.

Section 5 discusses FPGA development tool-qualification considerations, including general prospects for qualification and circumstances impeding tool qualification. Issues pertaining to qualifying tools for each step in the development process (netlist synthesis, functional simulation,

gate P&R, timing simulations, and verification testing) are presented in detail. Current practices regarding FPGA tool qualification and how designers cope with safety assurance for devices developed using unqualified tools are also discussed.

Section 6 presents 25 recommendations, in the areas of additional and improved design and verification activities, treatment of FPGA development using software considerations, guidance aimed at tool vendors, and dealing with activities that design engineers felt pose particularly high risks of defect injection. Conclusions appear in section 7.

1.4 TERMINOLOGY USED IN THIS REPORT

Although the tool-qualification issue in the SOW applies to PLDs in general, the dominant variety of PLD used in modern hardware design is the FPGA. This report will refer to FPGA configuration and tool issues; identical considerations apply to other varieties of PLDs having similar complexity and configuration properties. The traditional distinction between complex PLDs and FPGAs had in the past been that PLDs were based on macrocells implementing logic operations and containing on-chip non-volatile memory, whereas FPGAs had a larger number of available gates and were more suited to deeply layered logic. However, in modern devices, this distinction is blurred; for example, "third-generation" FPGAs including non-volatile memory have been available since 2007.

This report refers to "FPGA design" to mean design of the configuration that allows a specific FPGA to be used as a component of a system, meeting requirements imposed on that component. It is possible to interpret the phrase "FPGA design" as the process, done by the part manufacturer, of designing the generic FPGA chip itself, but that is never the meaning used in this report.

When the term "high-level design language" is used in the context of a means of expressing the requirements in a form suitable for creating the FPGA configuration, the word "high" refers to the fact that the language consists of logic statements, rather than the hardware gate configurations that implement those statements. This terminology should not be confused with the distinction between high-level and low-level requirements expressed in standards, including DO-178C [5].

The term "testing" encompasses a spectrum of activities, having in common only the idea that some implementation of intended functionality is checked, either by simulation or by execution in hardware, to determine whether the device as currently implemented performs as intended. Tests can range from early informal trials to determine how well the most basic functions work, to formal and repeatable tests that are part of a comprehensive verification suite. Because the term "testing" is vague, this report will eschew the use of that term except where it is intended to mean "any activity in that broad spectrum of functionality checking."

The term "verification," used per DO-254, refers to actions taken to prove that a product that allegedly meets a set of requirements actually does meet those requirements. Verification activities generally consist of applying a suite of tests, which will demonstrate that each functional requirement is met and that the product (in this case the FPGA) performs properly in the required environment(s).

The term "validation" is used to mean any actions taken to reduce the likelihood that the product conveyed to the verification testers contains any defects. This is relevant to a best practice among

hardware and software developers. It should be the intention of the design and implementation team that the product delivered to the verification testers passes all the tests as delivered (i.e., verification by independent testers should not be relied on as a step in the development process meant to shake out "remaining defects.") Realistically, the ideal of "defect-free before verification" may not be met. However, correction of defects found at the verification stage will be reflected in change management. The certifying authority or designated engineering representative (DER) can take the presence of a significant number of defects found at the verification stage as an indication of risk of further, undetected, defects. For developers to be confident that the product conveyed to the verification testers is clean of defects, trial activities and analyses must be part of the development process, and this research refers to such activities as "validation." In DO-254, note 2 to section 6.2 recognizes the value of "informal testing outside the documented verification process."

DO-254 (section 6.1) has a narrower definition of the "validation process," focusing on ensuring that the derived requirements are correct and complete with respect to the system requirements allocated to the FPGA or other hardware item. Although this is included in the definition of "validation" used here, tool qualification considerations have little connection to requirements validation. Therefore, most of the attention of this report concerns subsequent validation activities, such as simulation, which are directly affected by tool quality.

The term "synthesis" is used in this research to encompass all activities that contribute to the configuration of the FPGA (as opposed to validation and verification activities, and as opposed to development of requirements). The primary synthesis steps are creation of the netlist reflecting the intended logic, and P&R of the gates and other elements to implement that logical netlist. Of these, FPGA tools providers generally refer only to "synthesis" of the netlist.

In many places, this report uses phrases such as "at DAL-C" or "for Design Assurance Level (DAL)-B development." These should be taken as shortened ways of saying "for development of FPGAs intended to perform DAL C (or B or A) functions in systems requiring certification."

This report will refer to FPGA tools that lack acceptable evidence of proper behavior as "unproven tools." A tool may also be considered to be unproven if acceptable evidence exists, but is unavailable to the developer concerned with demonstrating safety assurance, or is not available in an acceptable form. This designation refers only to what the developer knows and can demonstrate about a tool. An unproven tool might in reality be high quality and error free, though it may be difficult to demonstrate this per safety assurance guidelines given the state of the available process and testing documentation.

1.5 RELATED ACTIVITIES AND DOCUMENTS

The following documents relate directly to the issues addressed herein and contribute to a baseline of current guidelines:

1. RTCA/DO-254 [2], "Design Assurance Guidelines for Airborne Electronic Hardware" (AEH). In particular, chapter 11 of DO-254 provides guidance for tool assessment and qualification.

2. RTCA/DO-330 [4], “Software Tool Qualification Considerations.” This provides guidance for tool assessment and qualification, which would apply if the statements configuring the FPGA were considered to be analogous to software.

3. RTCA/DO-178C 5], “Software Considerations in Airborne Systems and Equipment Certification.” This document was considerably revised after DO-254, and contains elements of updated process guidance that can be considered applicable to FPGA certification.

4. Certification Authorities Software Team (CAST) Position Paper CAST-27 [6], “Clarification on the use of RCTA Document DO-254 and EuroCAE Document ED-80, Design Assurance for AEH.”

5. CAST Position Paper CAST-28 [7], “Frequently Asked Questions on the use of RCTA Document DO-254 and EuroCAE Document ED-80, Design Assurance for AEH.”

In addition to these documents, the information in this report was gleaned from several combinations of FPGA design engineers, companies, and application realms. The core content may be considered in-depth knowledge of the design flow, testing history, and nature of defects discovered in the course of the two example FPGA solutions—the Interface Display Processor (IDP) and the Wideview Display Controller (WDC)—developed for aerospace applications. Additional experience from several other designs performed within the same division of the same company was also incorporated.

Discussions with product and technical personnel representing the various major FPGA vendors and development software vendors provided insight into capabilities and availability of artifacts and information relevant to qualification for the various available tool suites.

This in-depth core content was broadened by discussions with multiple engineers in different divisions of the same large company. Some of the FPGAs motivating these discussions were developed in the context of FAA and/or the European Aviation Safety Agency (EASA) safety assurance, providing some insight into practicalities concerning the current design process. To further broaden knowledge of ways in which FPGA tools might introduce defects into the design, discussions were had with eight additional FPGA designers from scientific institutions. Very similar concerns about reliable and correct performance exist in the applications discussed, although the consequence of defects would be failure or degradation of multi-million-dollar experiments, radiation safety issues, or medical treatment issues, rather than potential airborne safety concerns.

2. THE FPGA COMPONENT DESIGN PROCESS

The raw FPGA is turned into the component meeting the allocated-down system requirements by designing a configuration of gates and memory contents. Given the extremely high number of elements on the chip, the complexity of the allocated requirements, and the necessity of adhering to timing and routing rules, it has been impractical to design the configuration by dealing directly at the gate and cell level. To greatly reduce both the engineering work necessary and the likelihood of errors with respect to the intended functionality, designers use sophisticated FPGA design tools.

This section provides details about the process of developing complex PLD configurations at the level of sophistication that has been typical in leading-edge developments since 2010. The flow of activities that occur during the design and verification of a complex FPGA or other complex PLD is examined. The purpose is to identify FPGA development activities that rely on tools that would not ordinarily be created by the FPGA developer and might not undergo qualification. These activities may therefore be vulnerable to defects in those tools.

Detailed analysis of two design flows for FPGAs used in airborne applications are presented to provide experience-based justification for the identified sequence of activities and nature of tool usage. These analyses also reveal activities that are prone to introducing defects or to failing to detect existing defects. This information highlights areas in which tool qualification issues are particularly relevant.

2.1 DEVELOPMENT OF AN FPGA FOR USE IN AN AVIONICS SYSTEM

The process of developing FPGAs is to an extent dependent on the specific device being used, the tool set(s) available, the level and processes of testing required, and other considerations. However, certain aspects of development are generally present in any modern effort to design an FPGA for a specific function in an avionics system.

The starting point is the creation of a reasonably complete and correct set of high-level requirements and knowledge of the component's criticality in overall system performance. There is no particular "language" for expressing these high-level requirements, and up to this point one would be indifferent as to whether they were implemented by a Central Processing Unit (CPU) running some software module, by an FPGA, or by a large assortment of interconnected AND, OR, and NOT gates.

The developer will decompose these high-level requirements into low-level requirements, which are testable statements about the device functionality. Then the developer expresses these low-level requirements as statements in an HDL. Before doing that, the developer must choose some HDL to use. Possible choices are:

- The VHSIC Hardware Description Language (VHDL) family of languages, with statements oriented around basic low-level logic:
  - VHDL, a description language designed to standards IEEE 1164, IEEE 1076, and various other child standards including IEEE 1076.1, .2, .3, .4, and .6. VHDL statements can go all the way down to controlling a single gate, but also can be built using modules that provide higher level design concepts, analogous to statements in a typical programming language.
  - Altera Hardware Description Language supported by Quartus tools. Very similar to (and possibly just an extension of) VHDL.
  - Active-HDL™ by ALDEC® includes its own HDL.
- The Verilog family of languages, which are HDLs that are much like general-purpose coding languages, though they include low-level control capabilities:

- Verilog, a description language (standardized as IEEE 1364) with syntax much like that of the C programming language.
- Requirements may be expressed in C using ALDEC's CyberWorkBench™.

The designer will be entering these statements of low-level requirements in some sort of interactive development environment (IDE). This IDE is provided by a tool suite. The choice of tool suite restricts the choice of HDL language (VHDL or Verilog or others) to the language or languages supported by the IDE and tool suite. For example, VHDL is supported by Altera Quartus II, Xilinx Integrated System Environment (ISE), Synopsys Symplify, Mentor Graphics HDP Designer, and others.

In the course of decomposing the high-level requirements into low-level requirements that can be expressed as HDL statements, it is often discovered that the high-level requirements are flawed—inconsistent, ambiguous, or specifying something other than that which was intended. Feedback to the high-level requirements creation from this stage of the device design is optimal because there will be little wasted work. It is best practice to check requirements by detailed examination that includes review by the development team as early and thoroughly as possible, particularly because requirements-based testing will be an integral part of safety assurance.

2.2 THE ROLES AND NATURE OF FPGA DESIGN TOOLS

FPGA design tools translate relatively high-level specifications of the desired functionality (as expressed by a set of HDL statements) to a list of nets that connect gates and cells to implement that functionality. In this sense, the design tools are closely analogous to a compiler for software code; directly implementing a complicated module by writing assembler or machine code is a lot more error-prone than relying on a compiler to properly implement that same module expressed in a high-level language.

FPGA tool suites allow the design, automated analysis, simulation, and testing of electronic circuits using FPGAs or other PLDs. The design tools enable efficient FPGA development by providing tools necessary for:

- Synthesis—Creating the hardware configuration starting from requirements expressed in a HDL, which might be VHDL or Verilog. The register transfer language (RTL) code is automatically translated into a list of nets that connects gates or flip-flops together to provide a logic gate implementation equivalent to the RTL and high-level code. This list of nets is called a netlist.
- Place and route—The netlist elements are physically placed and mapped to the FPGA physical resources to create a file that can be downloaded onto the FPGA chip. The routing and placement of logic elements is done in such a way that the constraints required for proper chip operation are met. In principle, this could be done "by hand," but that would entail significant risk of violating subtle constraints, leading to unreliable performance.

The design tools also provide support for verification activities at several levels, from functional simulation of the high-level code down to FPGA-based logic analyzer capture (signal taps) during hardware execution on the actual device.

The roles played by these design tools are essential to development of FPGAs that behave as intended. For example, creation of the gate-level netlist by hand-translation of the HDL statements expressing the low-level requirements would introduce many more errors than might conceivably be made by the selected design tool. This is the case even if the design tool has not been qualified in the DO-254 sense. In virtually all professional-level FPGA design work, use of automated synthesis tools has supplanted manual routing of gates, because even when accompanied by careful reviews, routing by hand is associated with a higher risk of defects.

2.2.1 Sources of FPGA Design Tools

There are two principal sources of the design tools used in FPGA development. The first source is the vendor of the FPGA hardware. Three major vendors are Altera®, Microsemi®, and Xilinx®; within each vendor there are multiple design suites. Altera, for example, sells Quartus II® and Quartus Prime®, whereas Xilinx sells ISE and Vivado®. These tool suites consist of various separable components. For instance, Vivado has "High-Level Synthesis" (HLS) taking the design from requirement capture to netlist synthesis, "IP Integrator," "Logic Simulation," "Integrated Mixed Language Simulator," and "HLS," which is a verification accelerator.

The second type of tool source is a vendor of FPGA development software, providing common capabilities across devices from different hardware vendors. The two major common tool vendors are Aldec and Mentor Graphics; there are other smaller companies in this area. For instance, VHDL functional simulators are available in ALDEC Active-HDL, Cadence Incisive, MentorGraphics ModelSim, Synopsys VCS-MX, Xilinx Vivado, and various open products, such as Gnu High Definition Language. If the HDL selected is Verilog, simulators are available in ALDEC ActiveHDL/Rivera Pro, Cadence Incisive, Xilinx IDE, Mentor Graphics ModelSim, and Altera Quartus II. Many older commercial products and various open products can also perform Verilog simulation, though these are in general flawed or incomplete in some aspects of the full Verilog language.

Whereas the FPGA vendors invariably provide tools covering end-to-end synthesis, simulation, and testing, the development software vendors need not cover every aspect of the development cycle. For example, Aldec provides an ALINT™ tool to assist in vetting of VHDL statements, but does not provide place and route tools for the current generations of FPGAs (although Mentor graphics has two such tools: Olympus® and Nitro® SoC). However, Aldec and Mentor Graphics each provide simulation tools and other capabilities, and because these are the areas that distinguish the development software vendors from the FPGA vendors, these tools tend to be superior in some way to the analogous tools provided by Altera and Xilinx.

One example can illustrate this superiority. Aldec's CTS (Compliance Tool Set®) is aimed explicitly at DO-254 compliance, particularly at chapter 6.2, "Verification Process." CTS allows the developer to specify key aspects of the operating environment and inputs and outputs of the FPGA. Based on these features, a custom-made daughter board is created incorporating the target FPGA. This is connected to a standard peripheral component interconnect (PCI) motherboard card that resides in a PC and allows all of the test cases developed for functional testing and simulation to be applied to the actual configured FPGA. This helps to systematically provide comprehensive test coverage, a task which would be less well organized using the Quartus II tool set alone.

2.3 FPGA DESIGN ACTIVITIES

The process of designing an FPGA to meet a given set of requirements involves several steps. These go from development of requirements and assessment of the appropriate DAL to architecture and hardware decisions, to formulating low-level requirements, to creation of the FPGA configuration implementing those requirements (performing preliminary validation activities at some point), to thorough verification of the behavior of the configured device.

1. The intended purpose of the proposed FPGA within the system is specified, and the high-level requirements to fulfill that purpose are specified.

2. A specific device is chosen that is believed to have sufficient logic capability and speed performance to implement the requirements and environmental robustness suitable for the level of criticality associated with the purpose of the device.

3. The high-level requirements are decomposed into low-level requirements.

    a. High-level requirements are also used to develop a collection of test vectors (the testbench). These will be used for validation and verifications in steps 5, 6, 8, 9, 10, and 11. From a safety assurance viewpoint, it is preferable that the developers of the testbench be distinct from the FPGA design team so that the verifications may be creditable as independent assessment.

4. The low-level requirements are broken down into their implied state variables (which can be expressed in terms of memory cells and flip-flops) and logic statements. These are then stated in some form of HDL. Two examples of HDL are VHDL and Verilog. (standardized as IEEE 1364).

    a. Depending on the nature of the HDL used, and the nature of breakdown of requirements, the HDL statements expressing these “low-level requirements” might alternatively be more akin to design details (and simple groupings of these statements might be the low-level requirements in the DO-254 and DO-178 sense). Notwithstanding that distinction, this report generally refers to “HDL statements expressing low-level requirements.”
    b. As the low-level requirements are being formulated, issues of ambiguity or contradiction in high-level requirements may be fed back for refinement of step 1.
    c. A rough assessment of the total of low-level requirements may indicate that the chosen device has inadequate logic capability, necessitating refinement of step 2.

5. The HDL is processed in the context of a design tool that provides functional simulation. Issues uncovered in the course of functional simulation are addressed by refinement of step 3. In the example design flows that this report examines, functional simulation capability is provided by ALDEC HLS and emulation tools.

    Functional simulation relates to safety assurance in that it acts as a check/verification of the low-level requirements and the behavior of the netlist produced by the early synthesis step. In principle, a comprehensive test suite could be applied at this stage, satisfying some

aspects of design assurance by black-box testing. However, the comprehensive tests would still need to be reapplied at a later stage, after the design is realized in hardware.

6. A netlist is created, using a netlist synthesis tool that "compiles" the HDL statements into logic elements and connections. At this point, a more refined assessment is made to verify that the device logic capacity is adequate; if not, it is necessary to refine step 2.

    a. Functional simulation may be applied to the netlist, to the "source" HDP statements (as in step 5) or both.

7. The netlist is processed by a tool that does P&R, and obeys the timing and routing rules pertinent to the FPGA device being designed. This tool is generally provided by the FPGA manufacturer, or at least must work based on design-rule data provided by that manufacturer. In the example design flows examined in this report, the Altera Quartus II suite of tools is used for synthesis.

8. The specification of the device configuration at the gate and timing level, provided by the place and route tool, is fed back to the same set requirements based tests (the testbench) that had originally been used for functional simulation. This phase of testing includes timing information to verify that the resulting configured chip will perform properly per the requirements. The VHDL file incorporating this timing information is referred to as "back-annotation."

    a. Problems detected during this simulation or during analysis of the critical timings or resimulation of the back-annotated netlist require refinement of step 6, and may require refinement of the HDL specifications used at step 3.

9. Actual FPGA devices are configured according to the specification of the device configuration generated in step 7. These are used in hardware testing, assisted by design verification tools. In the example design flows examined in this report, components of the Altera Quartus II suite were used for downloading the configuration and for hardware-based testing.

    a. Problems detected during this testing require refinement of step 6 and may require refinement of the HDL specifications or the breakdown of high-level requirements performed at step 3.

10. FPGA hardware-based logic analyzer capabilities of the verification tools (signal taps) may be used to verify that the internal states of the device are as expected during the hardware testing. This may provide additional assurance that the collection of verification tests is comprehensive.

    a. Problems detected during this verification require refinement of step 6.

11. Analysis of signal timings may be performed to assure that the internal signals are within their specifications and will not exhibit random-interval misbehavior at a level low enough to evade detection during testing, yet high enough to be problematic during operation.

a. Problems detected during this verification require refinement of step 6.

12. Final steps in the development process involve responding to early manufacturing and quality-control phases, such as loading the working configuration onto multiple devices and stress testing them in various environments. No design tool qualification issues arise in the course of this last phase.

Note that whereas this process is conceptually a waterfall process, feedback from each of the simulation and testing steps may cause r-iteration (or refinement) of previous steps.

Steps 1, 2, 3, 4, 6, 7, and 9 (or the logical equivalents of these steps) are absolute necessities in the development process. Omitting step 5 is always unwise because errors detected at that level are much harder to detect and fix working with the netlist. Depending on time constraints, level of assurance needed, and other considerations, efforts on steps 8, 10, and 11 may be more or less comprehensive.

These design phases are shown in figure 1. The primary steps toward creating the FPGA configuration are shaded in pink; verification steps are in green. Validation steps (which provide important design feedback but would not typically be offered for safety assurance credit) are shown in yellow, with the key feedback paths shown as red dashed lines. The hexagonal boxes represent steps that rely on FPGA development or testing tools.

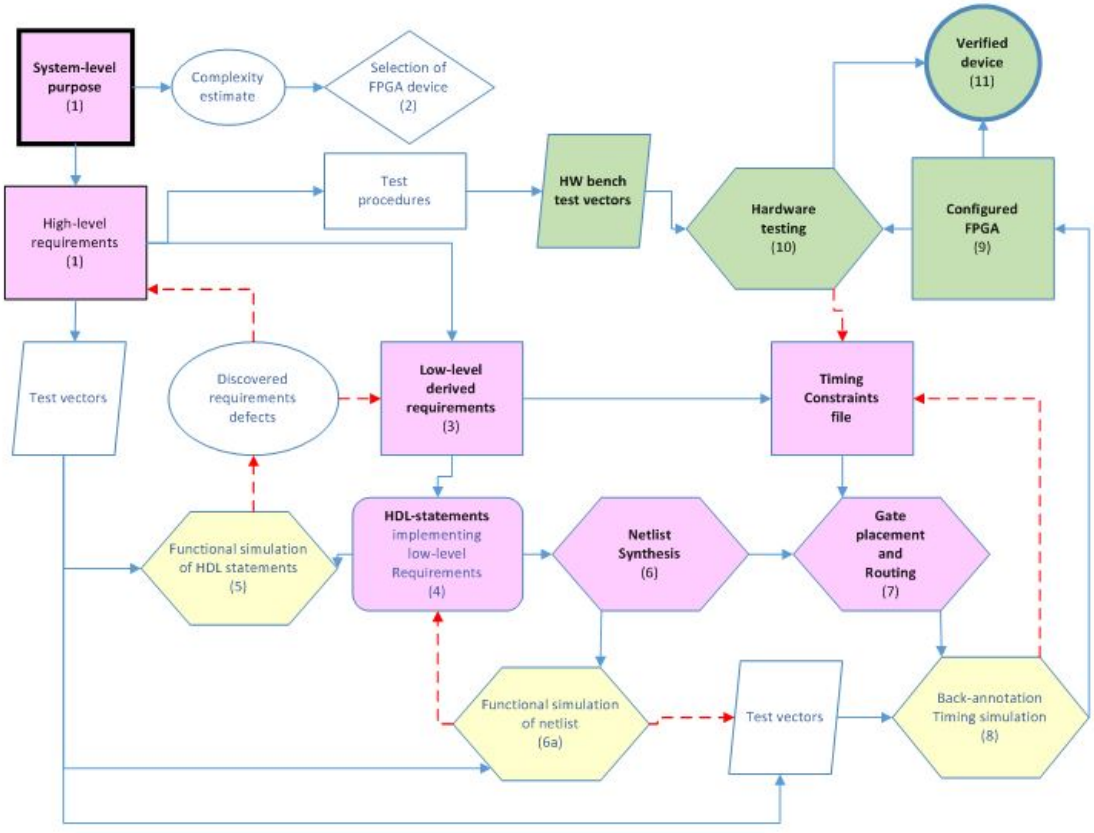

Figure 1. Generic flow of design for FPGA devices

Based on this process:

- Flaws in translating high-level requirements into HDL statements representing the low-level requirements should be detected at step 4. If the functional simulation performed is inadequate, these flaws will be detected by hardware testing in step 8 (at higher cost in total design time).
- Flaws in the translation of the high-level functionality specifications provided to the tool into correct logic net are detected at step 7. Therefore, it may be advantageous to use distinct toolsets for the place and route synthesis, and the gate and timing level simulation.
- Flaws in the P&R of logic elements, leading to incorrect or unreliable execution of the desired logic, are detected during testing at steps 7–8, and more stringently at step 9.

Tools specialized for FPGA design play an essential role in the creation and verification phases. The scope of this report is restricted to those phases of development. The FPGA development activities described here rely on tools that would not ordinarily themselves be created by the FPGA developer (and therefore may or may not be qualified). Those activities may be vulnerable to defects in the tools employed, raising issues of tool qualification.

2.3.1 Creation of the FPGA Configuration

In the design of an FPGA, the first step that involves tools that are specific to development of complex PLDs occurs when low-level requirements are captured in the form of statements in a hardware description language such as VHDL. Tools that may have played roles in earlier activities, such as organization and change management of requirements, have no special relationship to the development of complex PLDs and are out of scope for this report.

Creation of the FPGA proceeds from that point in the design until the configuration of the PLD exists and can be loaded onto the hardware to produce the intended device. For each activity in that string, the categories of tools involved and the vulnerability to potential tool defects can be identified.

1. Organization of (low-level) requirements for modularity. A skilled designer will organize requirements into groups, intending each group to be represented by one module in the hardware language. The designer will identify the state variables implied by the requirements, associating variables with the appropriate modules. The result of this activity is a splitting that respects principles that will reduce the likelihood of errors, such as isolation of concerns: each module ought to have one explainable purpose, and reliance on side effects via state variables that are controlled by other modules should be minimized. The same tools that assist this activity in software development (e.g., tools for creating unified modeling language [UML] diagrams) can be employed, though the use of such tools is by no means essential.

   In the context of FPGA development, organization of (low-level) requirements for modularity never relies on tools that produce statements in the hardware language (e.g., VHDL) automatically from UML diagrams. Therefore, tool qualification is not an issue for this activity.

2. Capture of the organized requirements into the hardware language. The result of this activity is a file or files containing statements specifying the detailed behavior of the intended device. This activity involves an editor, which may be:

    a. Part of the tool suite that includes subsequent processing of those captured statements.
    b. Tailored for PLD design and not part of the processing tool suite.
    c. A general-purpose editor.

    Although the editor tool is essential to this activity, the issues concerning potential tool defects are no different than for a code editor in the context of software development. In both contexts, safety assurance can proceed without needing qualification of editor tools.

3. Vetting of VHDL or Verilog statements. The result of this activity is confidence that the file of hardware statements produced is free of immediate "red flag" problems, which may increase the risk of errors.

    a. Checking against a list of common potential mistakes identified by community experience. Two examples are writing a statement that will lead to a transparent latch value when the intent is an edge-triggered D-type flip-flop, and causing incomplete assignment so that some value may not be well defined in every circumstance.
    b. Checking against organization and style rules.

    Although tools exist, akin to "lint" in the context of C software programming, which assist this activity, the vetting is ordinarily done by code inspection.

4. Creating the netlist from the hardware statements. The result of this step is the list of gates implementing the logic implied by the statements, which in turn were implied by the requirements. This activity and the later activity of going from the netlist to the chip configuration are the core of synthesis of the device. The essential VHDL or Verilog "compilation" tool is often, though not always, part of the same suite that includes the editor tool.

    a. This activity is synthesis of the netlist. In some contexts, the term "synthesis" is restricted to the creation of the netlist and excludes the activity of going from the netlist to the placement of gates and routine of connections on the actual FPGA. In this report, a more inclusive meaning is used of all steps proceeding from human-readable input to the downloadable configuration.
    b. The tool (or tools) employed in this activity directly determines the device behavior. Although there are later activities that will detect errors introduced by flaws in this synthesis step, it is natural to envision qualification of the tools performing this vital activity as part of the safety assurance process.
    Although no errors injected by the netlist synthesis tools were detected in the course of the two studied example design flows, in development (by the same teams) of devices dating back 5–10 years, 1–2 synthesis errors (correctable by reorganization

of hardware statements) had been detected in 50% of the projects. This indicates that correctness of netlist synthesis done by a nonqualified tool cannot be taken for granted.

c. The netlist is commonly produced as an Electronic Design Interchange Format (EDIF) file.

d. Netlist synthesis may occur before or after functional simulation.

5. Creating the configuration of the FPGA (gate P&R) from the netlist. The result of this is a downloadable configuration file that will turn the FPGA into a device implementing the intended functionality.

   a. The P&R tool processes the netlist, with an additional HDL file providing timing constraint information for this design.

   b. The tool employed in this activity must incorporate detailed knowledge to the constraints on timing, gate placement, and signal routing applicable to the specific device being configured.

   c. Configuration synthesis may occur before or after functional simulation and/or netlist simulation.

Capture of the organized requirements into the hardware language may be done within Active-HDL® by ALDEC, the editor in the Quartus II system by Altera (now the Qsys® product in the Quartus Pro system), HDL Designer, or HDL Author® by Mentor Graphics. Active-HDL has an advantage in that it provides a block diagram graphical level of abstraction; Quartus II has an advantage of easier integration of any Altera intellectual property (IP) precompiled blocks that need to be used. ALDEC's ALINT tool is available to assist in vetting of VHDL statements.

Netlist synthesis from the hardware statements may be done within the Quartus II or the Active-HDL environment or HDL Designer. Note that each of these is a monolithic integrated set of tools, and the capabilities that perform the various steps outlined in section 2.3.1, steps 1 to 5, are not in general identified with separate names. The tools tend to call netlist synthesis "compilation" of the VHDL "code." Because this step entails risks of tool flaws injecting behavior errors into the device, qualification of the component of the tool suite that does netlist synthesis would be relevant for safety assurance.

P&R is generally performed by a product obtained from the FPGA vendor (e.g., the Quartus II tool suite). This step entails risks injecting behavior errors into the device. Therefore, qualification of the component of the tool suite that does P&R is relevant for safety assurance. However, given the wide and ever-changing variety of FPGAs each manufacturer provides, it may be unrealistic to expect qualification for the P&R tools for each individual FPGA type.

No automated tools for equivalence comparison of independently produced EDIF files (netlists) exist at this time, so the EDIF comparison technique is not available as a validation or independent assessment verification possibility for safety assurance regarding correct netlist synthesis.

2.3.2 Preliminary Validation Activities

In the context of devices that require safety assurance using DO-254 or any similar standard, two distinct categories of test activities may be identified.

The first category involves tests applied for the purpose of checking the expected behavior of the device. These tests are applied to the results of preliminary synthesis steps. This type of testing is characterized by working to "shake out" the most likely areas for defects, by efforts to understand the details of behavior even when no defects are revealed, and by acceptance of the expectation that some problems will likely be identified and need to be rectified. This report refers to activities and tests in this category as "validation."

The second category comes after the device configuration has been completed and is thought, based on validation and/or other considerations, to be correct in all behavior. These are the verification activities outlined in section 2.2.3 of this report.

Whereas validation testing occurs in software and in non-PLD hardware development, in the FPGA context the validation activities are:

1. Using a functional simulator to validate the hardware language statements at the logical level. The result of this step is identification of errors in the hardware language statements. The essential functional simulation tool is often, though not always, part of the same suite that includes the editor tool.

   Functional simulation done in this manner, with a simulator working directly with the VHDL or Verilog statements, may occur before or after netlist synthesis. (Following a "test early, test often" development practice, it may be best if this is done before netlist synthesis.) Results of this functional simulation generally could not be presented for credit in a certification context because this is a simulation of an idealized device rather than the actual configured FPGA.
   Even when the module design and hardware statement generation has been done with considerable care, this validation step may reveal problems that need to be rectified. The earlier errors are caught and corrected, the smoother device creation will proceed. However, failure to detect any flaws during this activity does not prove that the design is error-free.

2. Using a functional simulator to validate the netlist at the logical level. The result of this step is identification of errors in the hardware language statements or in the synthesis of the netlist from those statements. The essential functional simulation tool is often, though not always, part of the same suite that includes the netlist creation tool.

   Functional simulation done in this manner occurs after netlist synthesis and need not work directly with the VHDL or Verilog statements. Netlist-level functional simulation can be done with or instead of hardware statement-level simulation. It is unlikely that results of this functional simulation would be presented for credit in a certification context because this is a simulation of an idealized device. This simulation does not take into account the P&R of logic elements. Even when the module design and hardware statement generation

has been done with considerable care, and sometimes even after pre-netlist simulation of the hardware statements, this validation step may reveal problems that need to be rectified.

Because there will always need to be comprehensive verification testing involving the actual device, in principle it is not necessary to qualify the functional logical simulation tool. However, because proper application of a correctly executing functional simulation can significantly reduce the likelihood of erroneous design, it may be desirable to provide an avenue for crediting suitable netlist-level functional simulation with a qualified tool.

3. Comparing EDIF files created from the hardware language statements, using two different netlist synthesis tools, to provide assurance of this step by independent assessment. Whereas this validation technique is possible in principle, it may require a sophisticated comparison tool because there is no guarantee that the order of appearance of logic elements must be identical in the two netlists.

Statement-level functional simulators appear in all the tool suites. Tools for creating a "test bench," a collection of inputs to feed to simulations and expected outputs, are major components of the ALDEC suite.

In practice, P&R synthesis activities often commence before the intended set of functional validation activities has been completed. Problem information derived from complications in creating a proper gate configuration, or from initial application of behavior testing, may provide valuable feedback. If these indicate that changes need to be made at the HDL level, those would generally correspond to changes in the low-level requirements driving the HDL statements. If such changes are made, then the requirements must be re-examined to ensure that the altered low-level requirements still represent a proper realization of the high-level requirements. When it is determined that the modifications of the HDL statements are satisfactory, the corrected HDL is used to synthesize a new netlist, and functional simulation is restarted.

A valuable validation technique is to use the P&R tool to generate a (possibly non-optimal) configuration and load that configuration onto an actual device. The requirements test suite (and typically additional tests and validation cases created by the developer) is run on this hardware, not for verification of a defect-free device, but for the purpose of early exposure of problems. Modern Altera FPGAs and the Quartus II tool suite allow examination of the internal signals on the FPGA by a signal tap, which is effectively a massive logic analyzer with timing and signal shape capabilities that can examine many key signals at once. Because this is such a powerful tool for detecting not only errors but places where slight timing disruptions might lead to errors, the step of detailed software simulation is omitted or curtailed in many development cycles when the hardware is running tests.

2.3.3 Verification Activities

One purpose of verification is to deliver results that can be presented for credit in a certification context. Whereas validation testing is useful any time something is learned about the device behavior, verification without recording results in an organized manner, or without considering the degree of completeness of coverage, would be inadequate for safety assurance purposes.

Verification activities are, by definition, activities performed with the intent of supplying artifacts for certification that the device behavior matches the designed behavior and meets the device requirements. For FPGA development, the verification activities are:

1. The synthesized P&R configuration is simulated at the timing and signal level. The simulation tool may be supplied by the FPGA manufacturer, but frequently it is provided by a third party (e.g., Aldec). From the safety assurance perspective, there are advantages to using a third-party timing and simulation tool, because that injects a degree of independent assessment of the performance of the placement tool and the simulation tool.

   This activity cannot be the exclusive means of verification. Aside from the fact that this does not directly test the actual hardware, simulation at this level is slow enough that it would be hard to provide comprehensive test coverage.

   A major reason for this simulation is analysis of signal timings to assure that internal signals are within their specifications and therefore will not lead to sporadic misbehavior during long-term operation. Because the types of errors exposed by this validation step might go undetected during comprehensive verification on actual hardware, there is reason for guidelines to encourage timing-level simulation. One way would be by providing an avenue for safety-assurance credit. This raises the issue of tool qualification for the simulation tool.
2. Full verification is performed by providing suitable test inputs to one or more configured FPGAs, and checking that the resulting outputs conform to expectations and requirements. This relies on setting up a circuit board (preferably the actual circuit that the FPGA will be part of) to supply the inputs for the various tests. Any circuit board that can provide all signals necessary for the verifications being performed is suitable for this purpose. If the verification uses a specialized environment other than the actual circuit that the FPGA will be part of, then additional evidence should be provided to show that the test inputs accurately represent the operating input characteristics.

   This activity can obtain additional verification information by exploiting FPGA hardware-based logic analyzer capabilities (signal taps) to verify that the internal states and key timings of the device are behaving as expected. If this additional information is obtained and used in safety assurance, then the issue of qualification of the tool providing signal taps arises.

Detailed timing-level simulation is skipped or curtailed in some FPGA developments. If that is the case, then it is important to use signal tap capabilities during activity hardware to verify that timings are as expected and within specifications.

Quartus II and ALDEC support gate and timing-level simulation of the FPGA configuration (Mentor Graphics makes a similar capability available). The “test bench” created for functional simulation can be augmented by input timing information written in the VHDL language. Ideally, this level of simulation would be applied to a comprehensive test suite appropriate for safety assurance; that ideal may be impractical if the simulation tool takes too long to perform an exhaustive test. Qualification of the detailed simulation tool would be relevant for safety assurance, allowing a basis for claiming certification credit for such testing.

The Quartus II system provides download capabilities and an excellent hardware-testing environment (step 10). In particular, this environment supports setting up signal tap capabilities exploiting "virtual logic analyzers" configured directly into the FPGA. This allows inspection of the actual electronic signals at critical points in the logic based on running the configured hardware on requirements-based test vectors. Creating the inputs to the FPGA starting from the test vectors already present in the test bench is a sound idea. However, the tool suites do not provide a means of automating this. Qualification of the signal tap aspects of such tools might be relevant for safety assurance.

2.3.4 Tools for Which Safety Assurance Issues Arise

As identified in sections 2.2.1 through 2.2.3, the suite of tools involved in all or most FPGA development projects consists of tools providing some or all of the following generic functionality:

- Capture of low-level requirements (design details) into VHDL or Verilog
- Detection of erroneous or risky constructs within the VHDL or Verilog code
- Transformation of the VHDL code into a netlist expressed as an EDIF file: "synthesis"
- Functional simulation of the VHDL code or of the netlist
- Production of a layout of gates, connections, and other logic elements: "place and route"
- Detailed (timing-aware) simulation of the placed and routed configuration
- Organization of verification tests and application in simulation in hardware contexts

Of these, the two varieties of synthesis tools are the most critical in terms of qualification or independent verification of their outputs. This is because defects in netlist generation or place and route will introduce defects in the FPGA configuration and, consequently, its performance. Whereas these defects may be exposed by verification testing, implementation by a cycle of "synthesize and introduce flaws, test, fix exposed problems, resynthesize" is not an optimal strategy in terms of safety assurance. Synthesis tools should be qualified (for DAL levels C and higher) or their output should be independently assessed.

The simulation and hardware test organization tools are moderately critical in terms of qualification. Flaws in simulation will frequently lead to false positives, in which a correctly functioning device appears to fail a requirement test. Multiple false positives may mask actual defects. Flaws in organizing the test cases and feeding them to the hardware or simulation cannot introduce defects into the device, but may allow existing defects to go undetected.

The tools for capture of low-level requirements and detection of dangerous constructs in the VHDL code are not critical in terms of qualification. The former is essentially an editor with various VHDL-specific convenience features; the latter is useful in eliminating certain potential errors made in the course of writing the VHDL code. Deficiencies in the behavior of these tools would neither introduce flaws nor allow defects to go undetected.

A library of IP blocks or cores, implementing common needs such as bus Direct Memory Access (DMA), may also be considered a design tool because it relieves the designers of the burden of coding the function covered by the block. IP blocks are as critical in terms of qualification or independent verification of their outputs as the other synthesis tools, because a flaw in an IP block will inject a functional defect into the FPGA performance. This report will indicate the (significant)

degree of risk associated with the use of unqualified library blocks, but not treat them as tools of primary concern because other ongoing FAA studies cover the IP libraries issue.

2.4 EXAMPLE DESIGN FLOWS

This section examines in detail the design flows of the WDC and IDP devices, using the generic activities framework described in sections 2.2.2 through 2.2.3 for uniform organization.

2.4.1 Design Flow for the WDC

Thales' Scorpion is a helmet-mounted display and cueing system in use on A-10, F-16, and other aircraft. The Wideview version of Scorpion uses a larger display unit (a 1024x768 personal display 175 [PD-175] collimated display) than the current production version, and provides an available binocular display of symbology. The larger display requires a more sophisticated controller, residing on the helmet with the display. The WDC is implemented as an FPGA.

2.4.1.1 Purpose of the WDC

In the Scorpion Helmet-Mounted Cueing system, an operational flight program (OFP) receives information and commands for the mission computer and from helmet-and aircraft-mounted inertial sensors, calculates and displays symbology and helmet-viewing information, and supplies video output to the display. The OFP runs on a module designated the Interface Control Unit (ICU). In Scorpion Wideview, the PD-175 display requires color-interlaced input providing detailed control of the liquid crystal on silicon (LCOS) at 480 frames per second. The ICU is not capable of supplying this with its video display generation and other duties. The WDC interfaces between the Scorpion ICU and the PD-175 display. The high-level system specifications also require capabilities of simple handling and format transformation of video streams to be overlaid on the primary display information and flexibility so that, with minor modifications, it can evolve for future uses involving different communication standards.

The primary responsibilities of the WDC component are to accept as input low-voltage differential signaling (LVDS) video signals generated by the ICU; perform color interlacing and intensity bit sequencing for each of the 1024x768 LCOS pixels in the PD-175 display; prepare the interlaced and bit-sequenced signals to drive the PD-175 at 480 frames per second; and deliver the signals to the PD-175 in a LVDS protocol.

Given the high-level system specifications and interface protocols, the starting point in development of the WDC chip was the detailed list of requirements for that component. If the creation of VHDL to configure the device were considered to be software and compliant with the DO-178C assurance standard then this detailed list would be the low-level requirements. For FPGA hardware the process is dictated by DO-254, which does not use the term "low-level requirements," but these detailed requirements are logically needed for the same reason that low-level software requirements would be needed in software.

There is a risk of flaws introduced by erroneous detailed requirements. This risk is mitigated by the practice of review of detailed requirements, as recommended by section 5.1.2 of DO-254. Because this is no different in concept or practice than requirement reviews in non-PLD context,

this report is not concerned with flaws introduced by erroneous detailed requirements or their remediation by reviews.

2.4.1.2 Initial Synthesis Phase

Although the generic list of synthesis activities begins with organizing the requirements for modularity, in this design, an earlier activity was deciding which of several available FPGA chips would be used to implement the WDC. The Altera Cyclone IV FPGA was selected. The relevant device characteristics include timing parameters sufficient for the input, processing, and output delivery; a large number of configurable gates and various types of memory cells adequate for the complexity of the mission of the WDC; and multiple high-speed links, each of which can be configured for any of a variety of communication standards.

The organization into modules, each of which would be responsible for some coherent set of requirements, was implemented using the graphical user interface of interconnecting VHDL modules provided by the ALDEC Active-HDL tool suite. This choice was motivated by the fact that the VHDL entry capability of the Quartus II tools provided by Altera does not support such an interface. The graphical interface provides hierarchical depictions of modules and their connectivity, so that the designer can easily "dig down" to lower levels of the hierarchy and ultimately to the VHDL instructions for the lowest level modules. The lowest-level modules in this development were designed to represent and implement a single low-level requirement.

Another advantage of Active-HDL was that the designers could add and isolate modules whose purposes were to assist in behavior analysis and debugging, rather than implementation of specific low-level requirements. Such "debug only" modules, which are possible though less natural within the Altera environment, later prove valuable for examining timings. Detailed analysis of simulated timings can provide confidence that the design will not fail in real operation because of signals that are not within specifications. (In practice, the developer checks against timings that are a bit tighter than the nominal device specifications to allow for fluctuations in the device or environment.)

The requirements grouped into each lowest-level module were then translated into VHDL statements implementing that module, which represent hardware-level instructions for gate and memory cell logic. The following risks of flaws were encountered in this step:

- Use of variables (which translate to memory cells or flip-flops) that were not previously established in a known state. This type of flaw is known generically as "uninitialized variables," and the VHDL entry tool does not automatically flag such flaws. Several instances of this issue were encountered in developing the WDC and uncovered during early validation steps.
- Coding in such a way as to create a "transparent latch" or D-type latch (which is not well defined during the "invalid" part of the clock cycle) in which the requirements imply that a flip-flop (which remains valid through the entire clock cycle, changing at rising clock edges only) is needed. In the WDC development, no flaws of this type were encountered. However, there is a large body of comments and advice on this topic (e.g., reference [8], indicating that it is a serious risk that grows as the complexity of devices increases).

- Erroneous translation of the requirement into the VHDL logic. For example, in some instances, intermediate results were combined in an incorrect manner or were not inverted when the requirement implied inversion was necessary. These flaws are later exposed during functional simulation of hardware language statements at the logical level. In experience spanning the creation of many FPGAs, designers find that "there are always errors exposed the first time you do this simulation," and in the case of the WDC, several flaws were indeed introduced during this step and revealed by the validation step of logical simulation. The risk of translation flaws is therefore significant, though the risk of an outright error surviving past the initial logical simulation is very low if proper consideration is given to make the validation test suite sufficiently comprehensive.
- Incorrect module interfaces and inter-module connectivity. Although simple conceptual errors concerning module output signals connecting to incorrect inputs of other modules are possible and completely analogous to erroneous translation of the requirements to a single module, another source of these errors related to tool usage is more important. In the hierarchical graphic tool for entering modules and connectivities, it is possible and easy to create a connection line that ends near to the (correct) input of another module. In such a case, the line may not register as a connection to the intended input, despite appearing to be correct under visual inspection. The graphical input tool does not catch such situations because it considers any output line leading to nothing on the page for the current module to be an output of the current module. This will show up on the next level up in the hierarchy as an extra output signal on the box representing this module; however, it is easy to overlook the superfluous output signal. This error is a form of erroneous translation of the requirement into the VHDL logic and merits attention for the purposes of this report because of the close association with the behavior of a key tool. In fact, when graphical tools are used, most of the errors that are "always exposed the first time you do simulation" may be of this type. That was the case in the development of the WDC. The risk of module connectivity flaws is therefore extremely high if this mode of expressing the requirements in VHDL has been used. The risk of an error surviving past the initial logical simulation may or may not be low, depending on whether the simulation tool (used later) flags undefined inputs in a way that becomes apparent to the validation tester.

The next step in the generic development process would be to vet the VHDL statements prior to "compilation." ALDEC provides ALINT, which is a tool distinct from Active-HDL, which can identify critical issues by checking coding style, and functional and structural problems. For example, this might well uncover uninitialized input that stems from failing to make an intended connection between modules. In the case of the WDC design, the VHDL statements were vetted by review of the modules generated. Some instances of errors in the graphical expression of the requirements were uncovered and corrected in this way, and this did not fundamentally reduce or eliminate the risk of such flaws.

For the WDC, the design flow added a step of compilation to a netlist prior to the hardware statement-level functional simulation. This had the effect of acting as a VHDL vetting tool in that it exposed a large fraction of the VHDL flaws, including in principle most of the module interconnection problems. However, most of the flaws were exposed via warnings and errors. The netlist compilation issues error messages when a VHDL syntax error prevents the compilation from understanding the statements and when inconsistencies among the VHDL statements lead to specifications that would lead to configurations that the compiling tool knows to be contrary to the

device design rules. The error messages were embedded in more than 10,000 warnings overall. Almost all of the warnings involved insubstantial or stylistic issues, although some warnings exposed issues with which the developer would need to deal.

The correct action in principle was recognized as elimination of every warning by addressing the issue causing the warning. This would have uncovered many of the flaws caught in functional simulation. The problem is that dealing with each of the warnings took as long as 2 minutes on average, so this approach would have required months of engineering time and delayed the project accordingly. Instead, all error messages were responded to, because development could not move forward until these were addressed. A typical class of error messages that were noted (and the flaws eliminated) were mix-ups in identifying an input by name in the lowest-level VHDL of a module, whereas the interconnecting modules generated by the graphic tool refers to it by a net number, or vice-versa. The warning messages were scanned to deal with those that looked substantial; the remainder was tolerated in the next iteration of this compilation.

2.4.1.3 Netlist Synthesis and Validation Activities

Functional simulation at the logical level came next. This simulation can be done in either ALDEC or the Quartus II environment, though for serious functional simulation at this level, ALDEC is the better choice. Therefore, the WDC was simulated at the logical level using a capability of Active-HDL. The Active-HDL supports implementation of a collection of test vectors (sequences of input values, with the required output behavior corresponding to each sequence) as a test bench that can be applied at this level of simulation and later at more detailed levels. An extensive test bench was created for the WDC based on the low-level requirements.

There was no review of this test bench with an eye toward ensuring that all requirements were covered. The test bench originated as a way to apply the validation step of logical simulation. The purpose of this validation step was to expose logic errors, which helps reduce the incidence of flaws in the final synthesis. Therefore, omissions in the test bench applied at this stage would neither inject flaws into the device behavior nor prevent detection during subsequent verification steps. DO-254 imposes no constraints on activities that can neither inject behavior flaws nor prevent detection of errors during verification. Therefore, DO-254 can be interpreted as not demanding complete coverage of all requirements in the test bench at this validation stage. However, it is often the case that the ultimate verification suite, which will be used in demonstrating DO-254-based safety assurance, evolves directly from the original test bench. In such cases, complete coverage of the requirements should be provided from the first validation stage onward. An appropriate means of achieving this is by a test suite/requirements trace review. This review has not yet been performed for the WDC. This is a deficiency that will need to be rectified as the usage of the WDC moves toward systems requiring FAA certification. A review of test coverage will be a necessary part of demonstrating that the system meets the safety assurance objectives.

As mentioned previously, a number of flaws were exposed the first time this simulation was run. Experience over a broad sampling of FPGA development projects indicates that it would be very unusual for the VHDL code to be logically correct on the first try. For the WDC, there were two iterations of fixing flaws exposed by functional simulation at the logic level before it was appropriate to move on to netlist synthesis.

When the design team was reasonably certain that the VHDL implementation of the requirements contained no more flaws that could be exposed and corrected by further simulation at the statement level, netlist synthesis was performed. In the case of the WDC, the "compiler" button in the Active-HDL tool was used. The VHDL files could have just as easily been transferred so that they were accessible from the Quartus II suite and compiled there. Both tools produce netlists in the EDIF, and the EDIF file produced by ALDEC tools may be used for netlist-level simulation by the Altera tools, and vice-versa.

- A possible source of injection of flaws is incorrect compilation for netlist synthesis. The probability of such flaws is low, though it would be preferable if there were some way to control and expose any such flaws before the netlist is used for P&R.

Netlist compilation yields a list of gates and connections that implements the hardware statements expressed in VHDL. Accurate synthesis of the netlist from the VHDL statements is fundamental to assurance of correct behavior, so it may be logical to demand verification of the correctness of netlist compilation, either by qualification of the synthesis tool or by assessment of the resulting netlist.

Currently, no "reverse compilation" tool is available to process a netlist and generate the corresponding VHDL code, which could then be compared with the original input VHDL. The most practical route to assessment of the netlist might be to create a second netlist using independent synthesis tools and compare the two netlists for equivalency. Unlike compilations of software code to form object files, netlist compilation is not expected to perform optimizations. Therefore, the EDIF files formed by two compilation tools should be equivalent in some sense, and as stated earlier, comparison of two EDIF files generated by two independent tools could act as independent assessment to demonstrate the correctness of the compilation step.

In practice, the two netlists are equivalent in behavior, though the ordering of elements and labeling of variables generally differ.

It is possible in principle to produce a canonical ordering of elements and convention for renaming variables, such that any EDIF file may algorithmically be transformed to an equivalent canonical form, and such that two EDIF files are equivalent if and only if their canonical forms are identical. At this date, there exist no available tools that would test whether two EDIF files (describing two netlists) are equivalent. If such a tool were available and qualified, the results of compilations by two different tool suites could be demonstrated to be equivalent. That could provide creditable independent assessment of the netlist compilation step during safety assurance analysis. This possibility would close a significant gap in the netlist generation aspect of safety assurance for FPGAs.

Having the netlist in the ALDEC tool suite, it was natural to perform netlist-level functional simulation of the WDC netlist within the Active-HDL environment. This allowed use of the same test bench as was used in the statement-level simulation. At this stage, the test vector implemented by the test bench was extended for slightly better coverage of the requirements; though, as previously stated, no formal demonstration of coverage completeness was attempted.

- A risk of failure to detect a flaw is associated with the possibility that the test bench does not cover all requirements. Whereas the netlist functional simulation is not the final verification, there is some likelihood that the verification test vectors will rely on the test bench. If any gaps exist, the verification may have the same gaps.

Some remaining flaws in the design were exposed by netlist simulation. Each of these was traced to a specific user error (often misconnecting module inputs or outputs, as discussed in section 2.4.1.2). It has been the experience of several FPGA development teams, designing a number of devices over the last decade or more, that the compilation of the netlist from VHDL statements does not itself introduce errors. This is understandable, in that the compilation process is not ambitious in terms of attempting optimizations. No evidence has been uncovered of errors being injected by the synthesis tools in generating the netlists for either of the two example designs presented in this section. In addition, several FPGA developers, outside the context of these example designs, have stated that they cannot point to any instance, in the last decade, of such error injection; this experience spans commercial and military aeronautics applications, and RF cavity control applications outside the aeronautics field. However, none of these developers, and particularly none of those familiar with DO-254-based safety assurance, would say that the lack of discovered error injection in their few projects constitutes a proof, or even strong evidence, that compilation of the netlist from VHDL statements never introduces errors. The authors know of no proof or body of accumulated experience that would demonstrate at a confidence level suitable for Design Assurance Level C or higher that such errors are never injected.

Qualification of a specific netlist synthesis tool is not a fully satisfactory answer to this issue, because the netlist elements available for one FPGA variety may differ versus another variety, so qualification artifacts used in the context of earlier projects employing the tool would not be relevant for projects using new FPGA varieties. Use of the synthesis tool would, in most projects, require that the tool be qualified from scratch. As discussed here in section 2.4.1.3, safety assurance by independent assessment of the synthesis step is a more promising approach.

2.4.1.4 Gate-Level Synthesis and Validation Phase

The next step in design of the WDC was creating the configuration of the FPGA (gate P&R). Although the Active-HDL suite might also have place and route capability, and the netlist was at that point in Active-HDL, it was more natural to trust the place and route engine in the Altera Quartus II tools because the Cyclone IV FPGA is provided by Altera. Although timing and placement/routing rules are provided to ALDEC by Altera for the purposes of configuration synthesis, if any discrepancy occurs, it is more likely that the Altera tools will have the newer intended rules. So it is more likely that the Altera version will be correct and up to date per information learned by experience with the FPGA chip.

Therefore, the EDIF file was exported from Active-HDL to Quartus II and used to create the configuration file. Creating the FPGA configuration also requires information about the timing of the inputs to the FPGA (including the clock input) and the timing requirements for the outputs. This timing constraint information is provided by the designer in the form of a VHDL file. For the WDC, this file was created in the Active-HDL VHDL editor.

The tool to process the netlist in Quartus II takes as input the EDIF netlist file and the VHDL timing file (which includes clock speed information), and produces the uploadable FPGA configuration file (extension.vho). The information provided in the timing file may also include environmental temperature specification. The device design rules will in general change as the temperature varies, and the netlist processing tool will place components in such a way as to implement the netlist under the combination of the timing constraints provided and the temperature range specified. Therefore, the processing of the netlist for an FPGA designed for use in aircraft should always be provided with the environmental temperature range specified in the device requirements.

- There is a risk of flaws being injected into the device behavior by incorrect implementation of the netlist logic at the stage of gate-level synthesis. The probability of such flaws is small, because the translation of the logic itself—ignoring possible timing and cross-talk and power considerations—is simple and direct.
- There is a risk of flaws being injected into the device behavior by element P&R that does not correctly follow the chip-specific timing, placement, and connectivity rules. The risk of this is very low, though not negligible, and the consequences serious because these can lead to sporadic misbehavior that might not be exposed even during verification testing.
- There is a risk of flaws being injected into the device behavior by element P&R using incomplete or incorrect chip-specific timing, placement, and connectivity rules. The risk of this is very low because the chip manufacturer has a deep understanding of the element behaviors and creates rather conservative rules. The consequences of incorrect P&R rules would be sporadic misbehavior.
- There is a risk that the correct timing, placement, and connectivity rules are applied, yet a specific FPGA chip misbehaves because of manufacturing defects. This possibility has little to do with the issue of tool qualification and will not be considered in further detail in this report.
- There is a risk of flaws being injected into the device behavior because of the designer supplying an incorrect timing specification VHDL file. Such incorrect timing may stem from simple user error in transcribing timing information derived from the intended device operating environment. However, it is more likely that the operating environment will differ from the specifications with which the designer is working.

  If the timing file is grossly incorrect, the tool producing the configuration will balk or report errors. However, slight flaws in the timing will allow the possibility of injecting sporadic misbehavior.

In the case of the WDC, there was considerable concern during physical device testing that the temperature increase within the device might be placing the design out of specification under certain heavy output conditions, causing the P&R to violate their rules. Although this has not been shown to be the case, it is certainly a possibility for other designs.

- There is risk that the presence of the device itself might alter the environmental conditions enough that the device configuration goes out of specifications. The probability is low and the consequences are sporadic misbehavior, which may be exposed in tests that exercise the device more heavily than normal operation.

The FPGA configuration file, after creation within Quartus II, was then moved back to the ALDEC environment and linked back in to get a simulation that includes detailed timing. This is an extremely useful validation, and could even be considered verification of correct behavior except, for two issues:

1. The tool performing the simulation is not qualified, and there is no easy way to obtain artifacts demonstrating a sufficient and relevant service history. Service history considerations are relevant because the tool in question is software, even though the tool is used in the design of a hardware component. Per DO-254, the lack of a demonstrated relevant service history means that results of the simulation produced by this tool are not creditable. Therefore, hardware testing will need to assume the full burden of verification of correct behavior.
2. Because the detailed timing-level simulation requires considerable CPU time, and because the most valuable part of the information is gained by looking at details of the internal timings, it was impractical to simulate more than a small fraction of the full test bench.

Detailed simulation of the WDC configuration occurred partially in tandem with hardware verification. It was useful in tracking down a small number of specific flaws, having been run on a selected fraction of the test vectors.

2.4.1.5 Behavior Verification

In parallel to the synthesis and validation activities, preparations were underway for hardware testing and verification. Quartus II has a framework for hardware testing, which includes the ability to extract, via signal taps, internal signals and waveforms. However, no facility is provided for supplying the inputs to the device or for extracting and checking the outputs. The developer is responsible for creating a test harness.

There are two reasonable approaches to creating such a test harness. The cleanest approach is to custom design a board that supplies input signals, allowing variations in the input logic values and detailed timing, and captures output signals. The testing board is controlled by a connection to a PC and also delivers the outputs (and output timing information) to the PC for checking against the correct results. This approach allows the systematic study of any flaws or near-flaws exposed, and it is reasonable to write software that will implement the test bench in the form of a suite of test vectors fed to the chip. A drawback is that thorough testing in the custom test harness still does not directly verify that the FPGA works properly in the actual operating environment. In particular, if some aspect of timing was incorrectly understood, then that error will propagate to these hardware tests so that the device may never be tested using the true timing. A second issue is that it costs considerable time and designer effort to develop a custom test harness, although the device manufacturer may make available a standardized development board, which can reduce the necessary effort. The ALDEC CTS® Compliance Tool Set includes development of a custom harness, but the price of that product was deemed out of range for the WDC development project.

The other approach is to place the FPGA being designed into some slice of the actual operating environment (the circuit board it will be part of). The decision between the two approaches is heavily influenced by the availability of the surrounding circuit board at the time the testing needs to commence. Relative to the custom-board approach, using the actual circuit board will faithfully

reproduce all timing and power issues, and this is much closer to testing in-flight operation. Because the circuit board needs to be developed anyway, this approach may also require less time and effort. The downside is that it may be awkward or impossible to feed the carefully developed test bench vectors into the circuit board environment, and it may be difficult to probe internal signals.

For the WDC, the actual circuit board was available, and that was the approach selected. During the initial stage of organization into modules, special decoupled modules were inserted for the purpose of allowing internal signal analysis. The Quartus II capabilities of interacting with such modules to produce a virtual internal logic analyzer were used during this "hardware-based simulation" and verification step. Technically, the configuration used when logic analysis was turned on differed from the minimal configuration that decouples all the debugging modules. The debugging modules do not affect overall board behavior and can remain connected.

Whereas a large set of test conditions was applied in the context of in-situ operation, the verification testing did not match the test bench used for validation. Flaws were exposed during initial verification. In some cases, examination of internal signals via signal taps revealed the source of the problem. In others, test vectors were added to the test bench and used in the timing-level simulation to study the problem in more detail. The most common source of error was associated with incorrect entries in the timing file, or unanticipated timing differences between the specification of the circuit board and the actualization of that specification.

Most flaws exposed could be fixed by modifying the timing file, producing a new configuration file from that and the EDIF netlist file, repeating some portion of the detailed simulation validation, and uploading the new configuration to the chip for reverification. At least one flaw necessitated going back further in the design, altering the VHDL statements to resolve a timing impasse. When this occurs, one must repeat the whole string of netlist synthesis, validation, and configuration creation. However, because the change is small and the test bench is already in place, this sequence is much quicker than it was in initial development.

The in-situ approach has one key risk:

- Testing in the actual operating circuit board may make implementation of a comprehensive test suite more difficult. This manifests as a risk of incomplete verification, leading to the possibility of failure to expose an existing flaw.

The custom circuit board approach has a different risk:

- If most of the verification testing occurs in a custom environment, then any flaws involving the difference between the actual operating environment and that custom environment will not be exposed.

At the cost of considerable extra design time, a combination of the two approaches might be suitable for high DAL verification. Another possible approach is to rely on subsequent black-box testing of the full system, with the designed FPGA in place, to provide overall safety assurance. This is the approach that will be used for the WDC chip.

2.4.2 Design Flow for the IDP

A commercial head-worn display system is being developed for military applications and commercial airline use in Europe and the United States. As such, developments are proceeding with attention to the FAA and EASA certification processes.

In this system, the IDP, implemented as an FPGA, interfaces between various aircraft-level busses and elements of the display system.

2.4.2.1 Purpose of the IDP

The IDP chip interfaces between the various busses and inputs specifying and controlling video at the aircraft level and the Smart Mobility Architecture (SMARC) card, which is a processor on the helmet-mounted display (e.g., IMX 6 or Intel Atom) that runs most of the OFP and contains graphics processing units (GPUs) that generate the displays. The intended functions of this single IDP component include:

- Performing all the interfacing in the aircraft Ethernet, Aeronautical Radio, Incorporated (ARINC)-429 bus, military Airlines Electronic Engineering Committee 818 video standard (including bi-directional optical link capability), and Society of Motion Picture & Television Engineers (SMPTE) 292 high-definition video standard.
- Performing certain critical portions of the OFP for symbology generation, based on input from aircraft.
- Accepting SMPTE-292 video input and other input standards for use in creation of the displayed video.
- Reformatting and manipulating input video signals and transmitting them (with overlaid symbology generated by its OFP functionality) to the SMARC display generator on the helmet.
- Sharing as feedback the LVDS signals generated by the SMARC, which are driving the physical displays, to monitor the displays for correctness.

The starting point in development of the IDP chip was the detailed list of requirements for that component. There is a risk of flaws introduced by erroneous low-level requirements. This risk is controlled by the usual process of a review of low-level requirements, as recommended by DO-254.

2.4.2.2 Initial Synthesis Phase

As was the case for the WDC chip described in section 2.3.1, an early activity was deciding which of several available FPGA chips would be used to implement the IDP. The IDP is implemented as an Altera Cyclone 5 SX FPGA. The device characteristics include:

- Approximately a dozen high-speed Gbit communication links (the actual number of links can be adjusted depending on overall configuration decisions). Each link can be configured for any of a variety of communication standards.
- A large number of configurable gates and various types of memory cells.

- Available IP packages for various capabilities including Nios “soft” processing cores and soft GPU substitutes (“soft” refers to components created by ordinary gate configuration). The Nios processor is a general-purpose Reduced Instruction Set Computer (RISC) processor.
- On-chip flash memory to enable rapid boot up.
- Two “hard” (built-in and hardwired, using faster or tighter circuitry than would be obtained by gate configuration) ARM9 processing cores.

The organization into modules, each of which would be responsible for some coherent set of requirements, was implemented using the VHDL editor in Altera’s Quartus II tool suite. This choice was motivated by anticipating the benefits of incorporating Altera-provided IP “soft cores” (configurations of gates and connections) for handling communication protocols. Although this incorporation can be done within other tool suites, Quartus II has a particularly clean and efficient interface for putting IP cores into the design.

In particular, several IP components needed to be injected:

- A Double Data Rate type-3 (DDR3) dynamic random access memory controller.
- A Nios processing core. The Nios core vendor asserts that the core is certifiable, meaning that artifacts will be made available to facilitate FAA certification. DO-254 compliance design services for the Nios core are offered by Hcell, a third-party vendor. It remains to be seen how successfully one can use these artifacts and services to achieve certification of the actual device.
- For some applications, though not the initial application, the soft GPU substitute will be needed.

The requirements were organized hierarchically, even though the entry tool for Quartus does not provide a graphical hierarchical view. Again, each lowest-level module implements a single low-level requirement. The risks of flaws encountered in this step match those for the WDC chip (and will not be repeated here), except:

- Incorrect module interfaces and inter-module connectivity. Because no graphic interface was used, there is no issue about apparent connections in a graphical view not being connected in the actual VHDL. This type of flaw is replaced by the (less common) issue of modules invoking lower-level modules with arguments in the wrong order, therefore “crossing signals” in their interconnections. The risk of module connectivity flaws is low. However, it is not as easy to catch a crossed signal by catching undefined inputs because, in the typical flaw, all inputs are defined.

The next step in the generic development process would be to vet the VHDL statements, prior to “compilation.” In the case of the IDP design, the VHDL statements were not vetted as a second pass review before moving to simulation and compilation.

For the IDP, the hardware statement level functional simulation was skipped, moving straight to the step of compilation to a netlist.

2.4.2.3 Netlist Synthesis and Validation Activities

Although early functional simulation was not done, development of a collection of test vectors (sequences of input values, with the required output behavior corresponding to each sequence) to be used as a test bench in the Active-HDL context began. An extensive test bench was created for the IDP based on the low-level requirements. It is felt that to demonstrate DO-254-based safety assurance, complete coverage should be provided from the first validation stage onward and that an appropriate means of achieving this is by a test suite/requirements trace review. Such a review, for the purpose of providing artifacts demonstrating that testing achieves complete requirements coverage, is planned as part of the safety assurance activities for the IDP.

Concurrently, netlist synthesis was performed. In the case of the IDP, this was done within the Quartus II suite.

The netlist was then moved into the ALDEC tool suite, as it was thought that the netlist-level functional simulation of the IDP netlist would be best done in the active-HDL environment. This allowed use of the same test bench being developed. No formal demonstration of requirement coverage completeness was attempted.

A number of flaws in the design were exposed by netlist simulation. Each of these was traced to a specific user error (which was then rectified). As is the case with the ALDEC tool for netlist compilation, it has been the experience over the last decade or more that the compilation of the netlist from VHDL statements does not itself introduce errors. Again, safety assurance by independent assessment of the synthesis step would be a useful approach. The fact that few if any errors have been observed to be injected by the netlist synthesis tools raises the possibility that these tools can be qualified or that service history credit can be claimed toward assurance of the tool outputs.

2.4.2.4 Gate-Level Synthesis and Validation Phase

The next step in design of the IDP was creating the configuration of the FPGA (gate P&R). As was the case for the WDC chip, the place and route engine in the Altera Quartus II tools was used because the Cyclone 5 SX FPGA is provided by Altera.

Creating the FPGA configuration also requires information about the timing of the inputs to the FPGA (including the clock input) and the timing requirements for the outputs. This timing constraint information is provided by the designer in the form of a VHDL file with timing constraints. For the IDP, this file was created in the Quartus II VHDL editor. The tool to process the netlist in Quartus II takes as input the EDIF netlist file and the VHDL timing file and produces the uploadable FPGA configuration file (extension .vho).

The FPGA configuration file, after creation within Quartus II, was then moved back to the ALDEC environment and linked back in to get a simulation that includes detailed timing. Detailed simulation of the IDP configuration revealed minor timing issues, generally involving forgetting to constrain module reset lines and subordinate clocks in the timing file given to the P&R tool. Simulation using the test bench was important. A key technique was to set windows around the timings at critical parts of the execution and to view the signals within those windows as waveforms.

In the case of the IDP, all flaws detected in detailed timing simulation could be traced to errors by the developer, either in logic or in supplying proper timing constraints.

2.4.2.5 Behavior Verification

In parallel to the synthesis and validation activities (the detailed timing simulations started earlier than was the case for the WDC chip, and were done more thoroughly), preparations were being made for hardware testing and verification. Quartus II has a fully featured framework for hardware testing, which includes the ability to extract, via "signal taps," internal signals and wave forms. However, no facility is provided for supplying the inputs to the device or for extracting and checking the outputs. The developer is responsible for creating a test harness. The approach of using the actual circuit board was selected for verification of the IDP. As was the case for the WDC, signal tap capabilities were exploited, although the logic analyzer functionality used was simpler because it did not rely on added debugging modules.

Whereas a large set of test conditions were applied in the context of in-situ operation, the verification testing did not match the test bench used for validation. Flaws were exposed during initial verification. In each case, examination of internal signals via signal taps revealed the source of the problem. The most common source of error was again associated with incorrect entries in the timing file or unanticipated timing differences between the specification of the circuit board and the actualization of that specification.

Some flaws necessitated going back the design, altering the VHDL statements. When this occurs, researchers must proceed through the whole string of netlist synthesis, validation, and configuration creation. This needed to be done twice for the IDP chip.

2.4.3 Additional Activities Driven By Safety Assurance

Additional steps will be needed to complete the demonstration of safety assurance for the WDC and IDP FPGAs in a DO-254 context. Although the specifics might be excessive in one aspect and deficient in another, this list will give an overall sense of the additional work needed and indicate where there are problematic issues.

- As noted in section 2.4.2.3, a review of the test bench will be needed to determine if it covers all the requirements, and it will need to be extended if it does not.
- It should be decided, and noted in the PHAC, how the risk of incorrect netlist synthesis will be dealt with. If the intent is to trust the correctness of the compilation step, it is necessary to determine what artifacts and evidence can be presented to justify this trust. If the plan is to provide assurance by independent assessment, a tool must be obtained or developed that would support equivalence comparison between two netlists.
- There needs to be a review of the timing file provided to the place and route tool. Because flaws in this timing file can inject sporadic misbehavior into the chip, this review should occur before creation of an uploadable configuration used for testing.
- There needs to be some means of running through the entire test bench (therefore providing good coverage in behavior verification) in the context of the actual environment, or a custom testing harness that will allow those test vectors to be inserted into the device. An

alternative to this would be to provide evidence of correct operation by many hours of flaw-free execution in a variety of contexts.

2.5 OBSERVATIONS BASED ON THE EXAMPLE DESIGN FLOWS

Common occurrences may be observed from analysis of the two example design flows. These can be used in formulating recommendations about how to deal with tool-related and other FPGA-specific issues when safety assurance is applied to complex PLD hardware.

2.5.1 The Need to Distinguish Individual Tools

A general observation is that although highly overlapping design teams used the same two overall tool sets (Quartus II and Active-HDL) to develop both the WDC and the IDP FPGAs, there were considerable differences in tool usage at individual steps of the development flow. At most of the design steps, the necessary development could not be done without assistance of an appropriate tool. Because both tool suites had the capability of providing that assistance, the developer had to choose which tool to use. This choice was made based on features, strengths, and weaknesses of the corresponding capability in each tool suite.

For example, the capture of the WDC requirements into the hardware language was done using the graphical module entry capability of Active-HDL, augmented by statement-level entry in an editor for the lowest-level modules. The same capture step for IDP was done in the Quartus II environment and editor, and this choice was motivated by the fact that the IDP would need to exploit a library of pre-coded (and heavily tested) modules that were available in Quartus II.

This was not the only instance of tool usage diverging, even though the same two overall tool sets were in play for both FPGA designs. The proper way to view the situation is that there were two collections of tools available, and each tool had a focus on one of the design steps. Despite the fact that the developer could invoke, for instance, the netlist compilation tool provided in the Quartus II environment by selecting an item from a menu and could invoke the place and route engine by selecting a different menu item in that same environment, these are logically two different tools. One can reasonably choose to apply the netlist compilation in Active-HDL and the place and route in Quartus II, and go back to Active-HDL for detailed simulation. Considering the tool suites as single tools, to be qualified or not, is a counterproductive viewpoint.

This leads to the conclusion that a major foundation in establishing organized criteria for FPGA design tool qualification will be identification of the individual tools that appear as capabilities in the various development environments or tool suites. This also holds for any process analogous to qualification in which properties of and experience with a tool, which may have been established by others rather than the current PLD developer, are used as creditable information in safety assessment of the results of that tool for the PLD being developed.

2.5.2 Error Types Observed

Flaws and error-detection failures associated with the use of tools can be classified into three types:

1. Problems caused because a properly used tool created an error, either injecting erroneous behavior into the FPGA or failing to detect an error that would have been exposed if the tool were working properly.
2. Problems that occur because of a combination of suboptimal tool interface and human usability, and imperfect attention to detail on the part of the user.
3. Problems that occur in steps using a tool but that have nothing to do with a flaw in the tool (i.e., they are problems caused by developer error).

Discussion of the specific errors that can occur, and their associated risks, appears in section 3 of this report.

An observation is that although it is possible to identify quite a few flaws that were injected, at least temporarily, during the design process, none of these were found to be due to improper performance of a properly used design tool. A similar statement holds for flaws that validation steps failed to expose: identifiable imperfections in the developers' activities were in each case found to be the cause, or at least a contributory factor. Therefore, although problems caused by tool failure are quite serious when they occur, they are at least rare enough that none were observed in the course of the studied design flows.

Problems caused by suboptimal tool interfaces were observed at a rate that indicates that they would be present in most complex PLD designs. These are issues such as a tool that failed to give an error message when encountering the use of initialized variables; tools that generated so many warnings, almost all of which were of no importance, that important warnings got buried in the noise; and subtle traps that increased the likelihood of user errors or decreased the ease of detection. A common example of the latter is erroneous connectivity in a graphical block entry tool, which looks to the designer as if the correct intermodule connections have been made. These errors are common enough that any tool qualification or qualification-like guidelines should mandate consideration of user interface pitfalls.

Several outright design errors were encountered in the course of development of each FPGA, and these were caught and remedied during validation and testing. Ultimately these were resolved, and the devices passed all requirement-driven tests. However, there are opportunities to strengthen the process by adding activities aimed at early detection of human (and tool-related) errors.

FPGA development has much in common with both software and hardware development. A "best practice" for reducing the likelihood of errors in software or hardware development outside the DO178-C and DO-254 contexts can be called "clean testing." The idea is that the device or code is developed and validated to the point that the development team is confident that no flaws remain. Then, and only then, a comprehensive requirements-driven test suite produced independently is applied. This presents a "defense in depth" against flaws emerging in the final product. If the device or code passes all tests on the first pass, then that fact lends confidence that few, if any, undetected flaws were injected.

In the context of FPGA development, this approach to testing becomes nonviable unless sufficiently comprehensive validation steps detect virtually every flaw. Therefore, defects in a tool may have safety implications even if the tool in question is used "only" for validation during development.

2.6 TOOL-RELATED CONCERNS IN CURRENT FPGA DESIGN PRACTICE

2.6.1 Tool Usage

Current practice commonly skirts the issue of qualification of the synthesis-type tools (netlist synthesis and place and route) by providing independent assessment of their outputs in the form of a combination of manual review of outputs and successful testing of the resulting configured FPGA. This combination is generally deemed acceptable by DERs. However, that acceptability may partially be influenced by the realization that tool qualification is at this time impractical.

Under strict interpretation of current guidelines, and assuming that because these tools are akin to software tools DO-330 is applicable, it is possible to argue that a tool need not be qualified if:

- The tool is used only to expose a defect that would otherwise have been detected in the course of comprehensive verification. In this report, this is referred to as a "validation" step. Early discovery of defects can make a large difference in the effort needed to produce a defect-free chip configuration. However, safety assurance based strictly on the DO-254 and DO-330 guidelines would not include demonstrating qualification for testing tools that are not involved in the verification of correct performance of the completed design.
    - As a practical matter, extra effort necessitated by failure of a validation step to detect flaws may have an indirect effect on safety assurance by consuming designers' time that might otherwise have been spent improving other implementation or verification activities.
- The tool is used in verification activities, and the device functionality is at design level C or below.
- The output of the tool is checked by comparison against the analogous output obtained using an independent tool.
- The tool or tool feature only provides assurance of coverage.

The conditions triggering the need for tool qualification are logically associated with "defense in depth" against undetected defects in the final design. In essence, two layers are that:

- The device implementation involving synthesis-type tools (which section 11.4.1 paragraph 4 of DO-254 refers to as "design tools") covers all requirements and injects no defects.
- The verification activities cover all requirements to verify all intended functionality and expose any defects present.

The practice of attempting to use intermediate validation steps to expose and correct defects so that the ultimate verification is not expected to find any defects adds a third layer at little net cost in effort because the time spent in validation is usually recouped in reduced effort of correcting defects at later stages.

The logical criteria for certainty, per DO-254, is that one of these layers is nominally assured to be free of flaws. That implies that the tools used for that phase require qualification, or their outputs must be independently assessed. It can be logically argued that for DAL-C development, it makes

as much sense to rely on qualified verification tools and to allow unqualified synthesis tools as it does to rely on qualified synthesis tools. It is recommended that this flexibility be considered. The ability to accept this inversion of qualifying verification in lieu of qualifying one or more implementation tools is particularly important because qualification of place and route tools, which are constantly evolving with the target FPGAs, appears to be unlikely.

However, the following must be kept in mind: at the implementation stage (using synthesis tools), it is natural to achieve complete coverage because the designer has to be going down the list of requirements to know what to implement. At the verification stage, it is relatively easy to skip some functionality, providing incomplete coverage. If this occurs, then the argument that it makes as much sense to qualify the verification tools as the synthesis tools no longer holds true. Reliance on verification tools must include a mechanism to ensure that coverage is complete. Under that circumstance, tool features that only provide assurance of coverage need to be qualified or in some other manner assured.

2.6.2 Introduction of Risk Associated With Use of Unqualified Tools

Possibly because FPGA development is not considered to be software development, demands for qualification of the design tools are generally less stringent than would be implied by DO-178C section 12.3 and DO-330. In some cases, this introduces little risk; this section delineates cases in which the risks of defects in the final product are affected.

The netlist synthesis tools provided by both the major chip vendors and the FPGA design software vendors are mature because they do not have to change in response to each new chip version. Therefore, designers typically rely on existing netlist synthesis tools without providing evidence of tool qualification. Because defects introduced by these synthesis tools are very rare or nonexistent, this situation does not introduce unacceptable risk. There is, however, a caveat: use of similar tools provided by minor vendors, or of trusted synthesis tools on operating systems other than their intended systems, does incur uncontrolled risk.

Functional netlist simulation tools are not essential to the safety assurance aspect of FPGA design. As discussed earlier, these are validation tools aimed at early detection of defects. Because the development process does not rely on functional simulation as a sole mechanism for fault detection, there is little risk associated with using unqualified functional simulators.

Place and route tools have both the highest safety assurance risks associated with use of unqualified tools and the most difficult path to qualification, at least under current guidelines. Instances of configurations that fail to meet one or more timing requirements, or contain other defects, have been observed. Whether this is due to incorrect application of the P&R tool, including incomplete timing-constraint specification or incorrect performance of the tool, is often a hard question. Companies developing FPGAs are not interested in reporting deficiencies occurring in their design projects in the open literature. Because problems at the place and route stage are very rarely “smoking guns” showing that a defect in the P&R engine introduced errors, these deficiencies also fail to show up on problem logs and open defect reports maintained by the tool manufacturer.

Nonetheless, incidents that raise suspicions of tool-induced place and route errors are seen more commonly in releases of new versions of the FPGA chips and particularly at a generation change

boundary. It would be desirable that early "shakeout" of possible errors would occur in less safety-critical contexts than FAA-certified aircraft. Tool qualification activities might promote that goal.

Timing-level detailed simulators are used both as validation tools and in verification. The designer will often want to offer the records of signal timings meeting correct expectations as verification evidence. This argues in the direction of needing qualification for such tools. However, the authors could find no evidence of risks of defects (more precisely, failure to detect defects) associated with the use of unqualified simulation tools and no evidence of "false positives" in which the tool incorrectly indicated an unacceptable timing situation. Such risks might be exposed if the detailed timing simulation were routinely applied to all verification tests, because there would be evidence of tests passing on actual hardware and failing in simulation, or vice-versa, and timing probes on the hardware verification tests could indicate erroneous simulation. The authors of this report could not uncover designs for which this level of replicated verification activity was performed and revealed tool problems.

The last concern involves the step of verification by applying a test suite covering all requirements to actual configured hardware. The tool issue here, if a tool is used to automate this testing, is the degree of trust that the full supplied suite of test vectors was actually applied, and that the results were properly compared to the expected results. Note that the principal risk associated with verification is that some requirements or corner-case conditions may be omitted. This is not primarily a tool-related risk.

3. CLASSES OF RISKS AND POTENTIAL DEFECTS

This section presents a spectrum of defects that might possibly be introduced by use of defective PLD development and simulation tools, and assesses the risks and impacts of these potential defects based on study of the example design flows and on extensive consultations with designers of many other FPGAs.

3.1 WAYS TO CLASSIFY ERRORS

Each error that might possibly be introduced by use of defective FPGA development or simulation tools may be described in terms of three characteristics:

1. Errors are introduced during various phases of the development process: synthesis (including the netlist creation and place-and-route steps), preliminary validation, or verification of behavior.
2. Some errors, if present, inject incorrect behavior into the device. Other errors cause failure to detect potential defects. Another type of tool error would mislead an assessment of the coverage achieved by verification. As an example of the latter, if an automated tool is used to apply the test-bench suite of cases to the configured FPGA, then erroneously and silently omitting some of those cases would lead to incomplete verification.
3. Errors can also be classified based on the identification of effects of the potential failure condition that the error can cause. In order of most serious to least, there may be: safety implications beyond the function of the device; misleading behavior of the device; major loss of device functionality; minor partial loss of functionality; or degradation of performance.

For failure modes that would lead to some of the devices exhibiting the flaw and others working properly, the identification of effects includes an assessment of probability of expression of the flaw. Similarly, for failure modes that would lead to transient or intermittent performance faults within a single instance of the device, identification of effects includes an assessment of frequency of exhibiting the fault.

The content of this section is gleaned from the study of the design flow of the IDP and the WDC, and from discussions with the broad spectrum of FPGA developers. The two usual categories of risks are those of injection of defects due to misperformance of a synthesis-type tool and the possibility of a verification tool failing to expose a defect that should have been detected by verification activities.

In addition, a third category is presented here: a tool may render it difficult to comply with some principle espoused in DO-254/DO-330. For example, the default operations of several P&R tools use pseudo-random simulated annealing algorithms, and unless attention is paid to avoid loss of information, the resulting configuration may not be precisely reproducible from the source HDL statements. Exact reproducibility is a valuable principle espoused in DO-254 and DO-330. Even if it is impractical to qualify the tool, and therefore independent assessment of the output in the context of the specific design will be performed, failure to adhere to this principle may introduce undesirable risks. In cases in which a tool makes it awkward to achieve desirable assurance characteristics, a path toward remedying that situation may be recommended.

3.2 RISKS OF DEFECT INJECTION

Defect injection by a synthesis tool is the most serious form of potential error. Two stages of synthesis are relevant: netlist creation, and placement/routing.

A distinct form of defect injection would be incorporation of flawed packaged IP blocks obtained from libraries. The degree of risk associated with the use of unqualified library IP blocks, which could potentially be flawed, is discussed in section 3.5.

3.2.1 Defect Injection in Netlist Synthesis

Almost all of the FPGA designers who contributed to this report had never encountered (in recent development projects) any instances in which the VHDL, Verilog, or C code (in which the netlist compilation tool allows starting at that higher language level) expressing the low-level requirements was compiled into an incorrect netlist. This statement should be qualified by the observation that several designers recalled instances of incorrect netlists being created because of flawed input. Some of these errors occurred because:

- It was easy to produce apparent routing connections in a block diagram entry tool that were not actual connections.
- Unintended error-prone constructs (which would have been caught by a VHDL style checking tool such as Aldec's ALINT) led to flawed netlists.
- Ordinary "coding errors" in the VHDL statements or C code.

An example of a very common error-prone construct is the use of uninitialized variables (signals that start out in an undefined state). Uninitialized variables do not always lead to problems; for instance, if the signal is set to a defined state at reset, and reset is done on power up, then the state is always defined by the time the signal is used. However, checking in each instance that no problem will be caused is more onerous than simply avoiding this error-prone construct. Whether or not DO-254 or other general standards require that all signals start out in a defined state, it is prudent to adopt project design standards that forbid uninitialized variables.

A fourth type of error introduced at this stage, writing VHDL that causes an unintended "transparent latch," is discussed at length in forums on the Internet, but none of the engineers who contributed to the report had personally encountered that type of error.

Avoidance of such flaws can be promoted by the use of a VHDL design rule-checking tool like Aldec's ALINT. FPGA designs for India's nuclear research program are required to "run everything through lint." Because this is by nature a validation step (helping to avoid the introduction of defects) as opposed to a verification step, it is unclear how far FAA guidance can go in encouraging the use of such tools. To the extent FPGA configuration design is treated as software design (see section 6.2), this form of validation becomes a natural component of assurance.

In contrast to netlist synthesis from VHDL statements, the direct use of block diagrams to express the required functionality does not inspire universal confidence. There were comments warning against using block diagrams in the Altera tools (e.g., "translation will break"). This may merely be a manifestation of the fact that it is easy to produce apparent routing connections. (Although that situation was described in the context of an Aldec tool, it may have been present for the Altera tool as well.) At any rate, full confidence in netlist synthesis from block diagram sources appears to be unjustified. This report recommends that the VHDL (or C) statements generated based on block diagrams and used for netlist synthesis be reviewed as source statements. For safety assurance assessment purposes, block diagrams should be treated as documentation of the intended behavior, rather than as the "source code."

In general, the netlist synthesis "tools do what you tell them to do." The confidence in proper behavior of netlist synthesis tools must be qualified in two ways:

- With respect to all the designers' experience, one example of improper synthesis (one functional error) was encountered; this was traced to the use of a 32-bit operating system when a 64-bit system was expected.
- In designs with clock domain crossings, instances of netlists implying metastability have been observed.

This situation of rare or non-existent tool-introduced defects, coupled with common designer-introduced errors, leads to a mindset of trusting the netlist compilation of VHDL (and for other applications, C) statements. From the viewpoint of DAL-B and DAL-A safety assurance, because the consequence of defect injection is serious, this report recommends that this trust be backed up either by qualifying the compilation tool or by independent assessment of the output EDIF file.

3.2.2 The Designer Error Issue

Although some types of errors encountered in section 3.1.1.1 are not, strictly speaking, tool-related, their consequences are the same as those of defects injected by a malfunctioning compilation tool. Most of the designers who contributed to this research affirmed that the FPGA "never works the first time." The fact that talented engineers regularly make these errors is related to the fact that expressing the intended function of the FPGA in the VHDL or similar language is more closely akin to software development than to complex hardware development.

DO-254 was written as a hardware development standard acknowledging some software-assurance concepts. DO-178C [3] was written and revised by authors more closely attuned to effective software development processes. Several ideas in DO-178C translate well to the FPGA development realm; this report identifies those and recommends that they be made applicable to FPGA safety assurance.

The fact that netlist synthesis will "do what you tell it to do" can cut both ways. VHDL can direct the production of transparent latches, clocks gated from flip-flops, and other design constructs that carry significant probabilities of erroneous behavior. State machines are clock sequential circuits, but the state can be based on conditions that are not mutually exclusive. If priorities are not specified, the netlist synthesis will still operate, but the behavior may not reflect appropriate priorities.

Asynchronous logic and multiple clock domains raise the likelihood of subtle design errors. A particular area of concern is the treatment of reset signals. These need to be replicated and aligned in each clock domain. If clocks are implemented using phase-locked loops, attention must be paid to the fact that the clock is undefined until the phase-locked loop is running, and this has implications concerning counters that are set coming out of reset. The area of clock and reset logic is the most important way in which FPGA design differs from an ordinary software effort. This report recommends that a clock-and-resets review be part of the FPGA development process.

3.2.3 Defect Injection in P&R

P&R is a synthesis step taking the netlist and various timing descriptions as input and specifying locations and interconnections among logic elements to create the FPGA configuration. The risks associated with P&R are fundamentally affected by the nature of what the P&R step is trying to do. P&R uses timing rules involving connection lengths and shapes, logic element required input timings, and guaranteed output time characteristics. The key is that logic states are captured in clocked registers; if the clock rate is too fast, some combination of output guarantees and signal delays will fail to meet the requirements for the corresponding input timings. Conceptually, P&R tries to arrange clocked registers implementing the netlist logic, optimizing for a fastest possible clock rate without violating those timing requirements.

The P&R engine "tries out" a fairly arbitrary set of placements and connections, evaluates the maximal working clock frequency, and then makes a series of perturbations on that layout that tend to improve the maximal clock frequency.

The matter is significantly complicated by the fact that typical FPGA designs incorporate multiple clock domains and that timing constraints on the overall input and output signals may be supplied.

Still, the idea of optimizing clock rates while respecting the timing rules and imposed timing constraints applies. ("Timing rules" include signal length rules, gate density rules, and any other gate-level design rules required for proper and deterministic chip operation). A key point is that the powerful optimization techniques being used involve guided pseudo-random perturbations in the layout. This has consequences that are discussed in section 3.4.2.

The prospects for qualification of a given P&R tool, either by process-based assurance or based on product experience data, are impacted by the facts that these tools employ very sophisticated algorithms and that the "target" FPGAs that need to be routed rapidly advance to new chip generations and versions. The vendors change generations every 3–4 years; internals can be completely different, tools change, and sometimes even VHDL macros change. The present discussion assumes that the P&R tool is not qualified; therefore, the possibility of tool defects should be considered.

The interaction of the FPGA design team with the P&R tool is an iterative one. The first netlist might be routed, and the clock timing might be unacceptably slow. In that case, the designer might introduce intermediate registers by breaking up complicated VHDL statements or blocks, allowing for faster clock speeds but introducing more cycles in the chip's throughput. The tool will sometimes indicate that a supplied timing constraint has been missed. The designer has to decide what to do in that case. The point is that there are ample opportunities for both tool-injected and designer-injected defects in the P&R step.

The types of risks associated with P&R, and therefore associated with the use of the (possibly unqualified) P&R tool, are:

- Incorrect functioning of the tool, leading to a layout that does not reflect the netlist.
- Incorrect application of the timing rules.
- Erroneous timing rules relative to the chip performance.
- Misunderstandings of subtleties when iterating the design.
- Designer-introduced defects due to incorrect/incomplete timing constraints.

The subtleties most likely to lead to defects are those involving multiple time domains.

In contrast to the situation for netlist synthesis, every designer who contributed to this research indicated a lack of trust in the results of P&R, and a need for simulation/early validation tests to expose defects. One designer had an example in which an oscilloscope signal exposed a timing defect that might or might not have been introduced by a tool flaw. Inspection of the most critical routes in the layout may be needed; at least one case was discussed in which the P&R engine was unable to satisfy the timing constraints at the originally specified clock speed. The tools can be configured to highlight paths that fail to meet constraints. This option should be located and used.

The statements made by the designers indicate that one rarely or never gets a proper layout on the first try. In particular, designs with multiple time domains require a great deal of care in the timing constraints supplied. A significant portion of the changes made to early layout attempts are modifications of the timing constraints file. This input, which is in the nature of a secondary source, has no direct counterpart in non-FPGA software development. The only mention of timing constraints in

DO-254 appears in section 2.1.3, bullet 1, in which they are listed as a derived requirement needed for hardware/software integration. This report recommends calling out the need to treat these constraints as configuration source material, on par with the VHDL statements and other source material used for synthesis of the device configuration.

With the exception of problems involving packaged IP blocks, no "smoking gun" instances of the place and route ("netlist compilation") step injecting defects were mentioned. In addition, the FPGA designers who contributed to this research expressed the likelihood that if other developers had experienced concrete instances of errors introduced by P&R tools, they would have heard of them. Nonetheless, in recognition that the P&R tools are more complex and perhaps less reliable than the netlist synthesis tools, this report recommends specifying a layout synthesis validation step or a more stringent verification involving checks of timing as part of FPGA safety assurance.

3.3 RISKS OF FAILURE TO DETECT DEFECTS

The tools essential to detect defects are:

- Netlist or VHDL-level functional simulation tools, which are used to validate the netlist synthesis step and the VHDL "coding" implementing the detailed requirements.
- Detailed timing-level simulation tools, which are used to validate the FPGA configuration generated by the place and route step, and to verify the design.
- Tools that measure timings during the running of the configured FPGA. These provide additional validation of the configuration.
- Tools that enable supply of sequences of signals to a configured FPGA and capture the resulting outputs. These, when applied to a set of tests providing sufficient coverage of requirements, provide verification of the performance of the configured FPGA.

Both the FPGA vendors and the software vendors provide netlist-level functional simulation tools. FPGA designers indicate that these are sufficiently mature that the likelihood of incorrect performance of the functional simulator is remote. A major reason is that these simulators need not change as the chip vendors move to the next FPGA version. The differentiators between the functional simulators provided by the FPGA vendors and those provided by FPGA development tool vendors lie primarily in external capabilities allowing the creation and organization of test suites. For example, Aldec provides a TestBench tool that organizes tests into VHDL Waveform and Vector exchange files [9], which can run these and record results in an organized way and can move these tests into actual hardware at a later phase using the CTS compliance toolset.

Defect detection at the netlist phase is not a matter of verification; it is a way to provide, to a later verification phase, a device with fewer defects. This supports the principle that the product delivered to the verification testers should be as clean of defects as possible, because each defect that verification needs to expose presents the risk of imperfect verification failing to expose that defect. At the netlist simulation phase, the most likely root cause of a failure to detect a defect that ought to have been exposed would be incomplete test coverage.

However, if netlist simulation is offered for verification credit (as opposed to only being used to present as clean a product as possible to the verification process) the issue of tool qualification may arise. These tools are good candidates for qualification because they are stable and hold long

service histories. A good alternative exists because this level of simulation is provided by both the chip vendor and the development software vendor, and the results can be compared. Therefore, this report presents no recommendations in the area of netlist-level simulation tools.

Timing-level simulation tools are used to validate the place and route activities, which have more error risk than the netlist synthesis. Moreover, the prospects of obtaining simulation tools with adequate qualification artifacts are diminished by the fact that at least part of the tool must be specific to the FPGA being simulated. FPGA designers who contributed to this research have not indicated experiencing any problems with incorrect behavior of detailed simulation tools. However, this is related to two general observations about such tools: They are not easy to use, and they run very slowly.

When entering a timing simulation using current tools, the signal names disappear, and some signals expected to be recognized will have been eliminated by optimization. P&R eliminating optimization can be rerun, which occasionally needs to be done to find timing problems, but the optimized P&R performed after attempting to correct the issues may reintroduce some of the problems.

In terms of the time taken for timing simulations, if the design process were to involve just one iteration of the place and route step, then it would be possible to arrange for many tests to be run in parallel on multiple powerful systems and to present simulation covering all the requirements for verification credit. In that case, tool qualification issues would be important at DAL-B and DAL-A. As discussed in section 3.1.1.3, the place and route step is iterative in nature, and in consequence, the use of detailed simulation tools is usually restricted to key tests exercising critical timing paths. Therefore, the fact that FPGA designers have not indicated experiencing any problems with incorrect behavior of detailed simulation tools cannot be taken as a strong indication that these tools are flawless. This report recommends that at DAL-B or DAL-A, if results of timing-level simulation using an unqualified tool are offered for verification credit, that independent assessment of those results be required. The independent assessment may be accomplished either by replicating the test using an independent simulator or by analysis comparing a portion of the results to observed hardware timings.

FPGA designers who contributed to this research expressed the concern that a significant number of timing constraints are needed to ensure defect-free operation, and there may be no adequate tool for automated checking that all those constraints are met. Altera’s TimeQuest tool and Xilinx’s Trace address this need to some extent.

Tools that measure timings during the running of the configured FPGA are inherently diagnostic tools. Design engineers find these tools to be very reliable. Because a defect in a timing measurement capability will not allow functional defects to go undetected, this report does not make recommendations regarding use of unqualified tools of this nature.

Tools that enable supply of sequences of signals to a configured FPGA and capture the resulting outputs are inherently going to be used to provide creditable verification of the performance of the configured FPGA. The potential mode of tool-induced error would be for the tool to fail to present (or to inaccurately present) one or more of the test cases. This would be a serious error because it

would result in the test coverage omitting some requirement, thus allowing any injected defect affecting only that requirement to pass undetected.

Among the projects studied, a tool like CTS would be a luxury. Instead, verification is done either by inserting the FPGA into its actual environment and providing conditions that exercise all the requirements or by creating a system to feed specific tests into the configured FPGA. Both of these methods are laborious and carry at least as great a risk of requirement coverage gaps as would the tool-assisted verification. Nonetheless, for safety assurance at DAL-B and DAL-A, it is clear that such a tool needs to be qualified if a coverage gap could potentially occur as a result of a tool defect. Fortunately, the vendors of such tools intend them to be used for DO-254 compliance (see, for example, https://www.Aldec.com/en/solutions/do_254_compliance) and may provide (for a cost) sufficient artifacts/service history data to support tool qualification. This report recommends that using an unqualified tool of this nature as the sole means of achieving test coverage should be considered unsuitable for DAL-B and DAL-A.

3.4 PREVENTION OF DESIRABLE ASSURANCE CHARACTERISTICS

The use of a tool should not make it awkward or impossible to retain a characteristic of the developed product that was desirable from the viewpoint of assurance of correct operation. For FPGA development, this issue may arise when the place and route tool is used to create the FPGA configuration. The desirable characteristic is exact reproducibility.

3.4.1 Exact Reproducibility

The process of configuring an FPGA is closely akin to the development of software. This report suggests, in section 5.1, that many tool-qualification concerns addressed by DO-330 and software concerns addressed by DO-178C apply equally well to FPGA development. One such concern is exact reproducibility of the "build" (in this case the FPGA configuration) from sources.

DO-330, in the last sentence of section 2.0, asserts that "… for a tool whose output is part of the software (sic) and thus could insert an error, it should be demonstrated that qualified tool functions produce the same output for the same input data when operating in the same environment." The role of the synthesis tools (including place and route) is comparable to the role of a compiler, which if defective could insert an error into the object executable code.

This imposes the same burden on FPGA development (treated as software for this purpose) as DO-178C imposes on ordinary software: it must be possible to start from the statements (VHDL in this case) and reproduce the identical output (FPGA configuration) at a later date. This characteristic is highly desirable from a long-term maintenance perspective. It is what allows minor defects discovered in an otherwise mature product to be repaired, and although the full verification must be repeated, the confidence gained by years of correct operation in all other cases can justifiably carry over.

3.4.2 Reproducibility of P&R

Most of the FPGA design engineers who contributed to this research pointed out that, on repeating the place and route step multiple times, the configurations created generally will not be identical. Because the job of the P&R engine is to produce a good (fast clock cycle allowed) implementation

of the netlist, each configuration is a solution to the problem, and the engineer is free to choose the best one. Sometimes the P&R engine reports that timing constraints could not be met, or were barely met; in those cases, rerunning P&R may result in a better configuration, meeting all the constraints. The obvious worry is that the process of creating the configuration may not be reproducible. It is necessary for the following analysis to understand why the configurations are different.

For FPGAs as large and complex as those currently in use in avionics and other applications, there is a staggering number of possible gate placements that would logically implement the netlist provided to the place and route tool. Most of those placements would fail to respect the constraints provided by the designer, or would require very slow clocking to meet the timing rules. The optimization of placement and routine is a difficult minimization problem in a space with a very large number of variables.

The details of the minimization algorithm used are proprietary to each tool vendor, but a good deal can be deduced about the general activities of the place and route engine. Techniques developed for obtaining good (though not necessarily absolutely optimal) solutions to such minimization problems are dominated by "Monte Carlo" methods, in which a fairly good solution is perturbed in a well-chosen but pseudorandom manner. The resulting new solution is kept if it improves the quality of the minimization, or otherwise the change is kept with some probability that drops to zero as the minimization quality decreases. The willingness to accept certain "counterproductive" steps allows the search to avoid being trapped in local minima. The specific algorithms used are almost surely in the simulated annealing family (in which the willingness to take a step backward is slowly diminished, emulating a liquid that is slowly cooled) or the genetic algorithm family (in which two good solutions are "mated," retaining large parts of each).

The point is that some randomness is involved, although this is the pseudo-randomness associated with a software random number generator. A pseudorandom number generator starts with a seed obtained from the user's input, a system clock register, or a fixed value, and repeatedly performs transformations on its state, each of which allows it to deliver one number. Generators will deliver the same sequence given identical seeds (generators introducing true randomness at each stage are not normally used). If identical netlists and timing constraint files and directives such as the amount of effort to expend in optimization are delivered to the P&R engine, then if the random seeds used are also identical, the identical configuration will emerge. If the random seeds are different, then very different configurations may emerge, and one may be slightly "better" than the other.

It is possible, at least using the P&R engine in the Quartus II tools set, to control and save the random seed such that the same configuration can repeatedly be obtained. Note that if anything about the input changes, including the amount of effort assigned to optimization, then the meanings of each random number change so that a very different configuration may be obtained, even when using the same seed. Most engineers consulted were aware that non-reproducibility was undesirable, and several warned that it would occur. However, the knowledge of the techniques required to ensure reproducibility is not widespread, even among experienced FPGA design engineers.

From this situation, it can be deduced that the interfaces or documentation of the various P&R engines do not make control of the random seed an easy feature to find. It is possible to tell whether

two configurations intended to be identical are in fact identical without laboriously simulating and comparing detailed timings. Each configuration has a cyclic redundancy code checksum, and if the checksums are equal, the configurations can be presumed to be identical.

Reproducibility would be a requirement for software developed to DAL-C and above. This report recommends that for FPGAs developed to DAL-C and higher, prior to verification, the FPGA configuration be built twice and checked to assure reproducibility.

3.5 DEFECT INJECTION ASSOCIATED WITH IP BLOCKS

Safety assurance issues concerning the use of IP blocks or cores, which are developed for general use by a team other than the FPGA developers for the project at hand, have a good deal in common with assurance issues regarding design tools. As is the case for synthesis tools, the IP block relieves the designers of the burden of detailed implementation of the functions covered by the IP block. Just as a malfunctioning synthesis tool may inject a defect into the FPGA behavior, a flaw in an IP block may lead to functional misbehavior. In addition, the same obstacles that may present difficulties in qualifying general-use tools may hinder the safety assurance process regarding IP blocks. These obstacles include difficulty in obtaining design artifacts for a tool or IP block that was not originally designed for DO-254 or DO-178C/DO-330 based qualification, and demonstrating relevance of existing product service experience.

DO-254 does not directly address the issues associated with the use of IP blocks, but because these issues are similar to issues arising when synthesis tools are used, the concepts in Chapter 11 (regarding tool qualification) of the guideline document are the most closely applicable to the safety assurance of systems making use of IP blocks. Therefore, this report treats prepackaged IP blocks available for general use as needing qualification or other demonstration of safety assurance in the same manner as synthesis design tools.

The one place where FPGA designers consulted for this research consistently mistrusted their design tools was in the use of prepackaged IP blocks. These are predeveloped subconfigurations of gates that may be placed at various spots in the configuration to provide a bus controller, protocol matching, a DMA controller, or even an entire processor. Every engineer warned of the need for explicit simulation of this sort of blocks to verify that they perform correctly (or to expose defects that must be corrected by modifying the source). In some cases, the internal netlist details are not exposed and functional simulation is more difficult. For IP with opaque internals, the issue arises that if testing exposes a defect, there is no obvious was to deal with and correct the problem.

Defects were indeed injected by reliance on such blocks. A particular example is that a DMA controller provided by one of the vendors contained a flaw. Although this was rectified after six months, the response of the vendor to the original report did not inspire confidence; the designer ultimately “reinvented the wheel,” writing his own controller. Other problematic blocks were protocol matching blocks, other than those involving the native protocol of the FPGA.

The vendors do provide defect lists, and the authors were able to follow the deficiency reports and ultimate resolutions of some of the defects the designers had mentioned. However, the authors found no instances of entries on these publicly available lists that referred to open (unfixed) flaws. This is a strong indication that the vendors (at least those vendors that part of this research

examined) reveal known problems only after they can provide the reassurance of a fix. If this policy is true, the service experience data does not meet the guidelines set in sections 11.3 and 11.4 of DO-254.

"Public" IP (blocks for which the VHDL source is widely available and source documentation is available) is regarded as providing a good starting point for implementation of the intended functionality, but it is recommended that the FPGA designer take control of the VHDL code, treating the IP code as advisory, and perform validation checks accordingly.

3.5.1 Qualification of NIOS II Cores

A NIOS II Core is an IP block, designed for the Altera family of FPGAs, which implements a processor by configuring a subset of the FPGA's gates. Similarly, the MicroBlaze soft microprocessor core is designed for Xilinx FPGAs, and DirectCore is designed for Microsemi FPGAs. A mature program, provided by a third-party vendor (Hcell), makes available design documentation artifacts aimed at DO-254-based assurance of systems, including NIOS II_SC (safety critical) cores. The intent of these qualification artifacts, which are made available for purchase by the vendor of the IP block, is to demonstrate compliance of the design process so that systems using these cores need little additional safety assurance activities related to the functioning of the cores. This is the same concept as would apply to vendor-supplied tools, for which qualification artifact packages may be available. Indeed, in some instances, vendors use the terminology "qualification" of IP blocks, as if they were in the same category as tools.

If these artifacts are relevant and valid, in the sense that they demonstrate that the design process for the IP block complied with the appropriate means of compliance, then safety assurance regarding the use of the IP core block can be streamlined. This is analogous to the use of tool design artifacts to qualify a synthesis tool, and the terminology "qualification" artifacts for NIOS cores is applied. It is sensible to consider such IP blocks to be in the same category as development tools, and to apply similar qualification guidelines as would be applied to other tools. Even if NIOS II cores are treated as tools in this way, the safety assurance aspects of the use of qualified NIOS II cores is out of scope for this report, which deals primarily with tools that cannot be qualified in the usual manner.

3.6 STATE MACHINE TOOLS

Vendor-provided tools amounting to IP blocks setting up state machines are an attractive feature. However, multiple FPGA designers consulted on this research have experienced "significantly incorrect VHDL results" associated with reliance on such tools. This report recommends encouraging or requiring analytic verification of the VHDL generated when state machine blocks are auto-generated.

3.7 TABLE OF VULNERABILITIES

The risks of injecting defects and failure to detect defects are summarized in table 1. Note that, although for a defect to emerge in the finished product, there must be injection and failure to detect, safety assurance implies suppression of both types of vulnerabilities. In particular, injection of defects is in itself unacceptable at DAL-B and DAL-A.

In this table, the first column indicates whether this is a risk of defect injection (I) or failure to detect (D). The level of risk is purely an assessment of how likely the given vulnerability is to occur, without considering the consequences if it does occur. The last column describes the severity of consequence: a high-severity risk of defect injection coupled with a high probability of detection failure on that same flaw will mean a likely defect in the system.

Defects that are purely consequences of errors on the part of the designer are included in this table to put the potential tool-introduced defect risks into perspective. These are indicated by a (U) on the level of risk or a (U/I) if the tool interface contributes significantly to the risk.

Table 1. Tool-related vulnerabilities to injected or uncaught defects

I/D	Design phase	Vulnerability	Level of risk	Severity
I	Netlist Synthesis	Poor modular organization	High (U/I)	Low
I	Netlist Synthesis	False routing connections	Med-high (U/I)	High
I	Netlist Synthesis	Error-prone constructs (no ALINT used)	Medium (U)	Low-med
I	Netlist Synthesis	Error-prone constructs (ALINT used)	Low (U)	Low-med
I	Netlist Synthesis	Uninitialized variables (no ALINT used)	Medium (U)	Medium
I	Netlist Synthesis	Mis-transcription of requirements	Medium (U)	Low-med
I	Netlist Synthesis	Transparent latch	Low (U)	High
I	Netlist Synthesis	Meta-stable states	Medium (U)	High
I	Netlist Synthesis	Incorrect VHDL compilation	Very low	Very high
D	Netlist Validation	Incorrect functional simulation	Very low	Med-high
I	Netlist Synthesis	Partially uninitialized variables	Low	Med-high
I	Place and Route	Incorrect timing constraints	Medium (U)	High
I	Place and Route	Incomplete timing constraints	Medium (U)	High
I	Place and Route	Multiple time domain problems	Med-high (U)	High
I	Place and Route	Erroneous timing rules	Low-med	High
I	Place and Route	Incorrect application of timing rules	Low	High
I	Place and Route	Layout does not reflect netlist	Very low	Very high
D	Detailed Sim	Erroneous timing rules	Low-med	High
D	Detailed Sim	Simulation tool error	Low	Medium
D	P&R Validation	Timing measurement tool flaws	Very low	Low-med
D	P&R Validation	Incomplete set of timing checks	High (U)	Medium
D	P&R Validation	Erroneous checking of constraints	Med-high	Medium
D	Verification	Incomplete test coverage	High (U)	Medium
D	Verification	Inaccurate test environment	Medium (U)	High
D	Verification	Erroneous test envir. implementation	Low-med	High
D	Verification	Incorrect supplying of test vectors	Very low (tool)	Med-high
D	Verification	Incorrect supplying of test vectors	Med-high (hand)	Med-high

In table 1, a very low risk level is assigned if none of the FPGA designers contributing information have experienced or heard of others experiencing the indicated defect mode, and all indicated expectations confirm reliance on correct performance. A low risk indicates that no example of actually encountering the defect mode was reported, but with a lesser degree of confidence and a greater propensity to cross-check the tool output before moving to the next design step. For example, the low (rather than very low) risk of incorrect application of timing rules, and the low-medium risk of the rules themselves being erroneous for the device being configured, reflect the fact that these activities change in essential ways for each new generation or version of FPGA chip.

Vulnerabilities at medium risk are those that are encountered occasionally in an appreciable fraction of FPGA design projects, but not with sufficient frequency to become a major source of problems. High-risk vulnerabilities are common occurrences. For FPGAs developed with the intent of demonstrating DO-254-based safety assurance for credit toward FAA or EASA certification, the high risk of incomplete test coverage is in practice mitigated by reviews of the suite of tests. Even so, if the large collection of test vectors providing that coverage are set up completely by hand, there is some risk that certain tests will fail to evaluate what they are intended to cover.

The last column in table 1 indicates severity of the potential consequence of each vulnerability. This is to be understood in the context of the "do it right, then verify" mode of development: a high severity of consequence indicates that when the corresponding vulnerability occurs, there is a significant likelihood of either defect injection or failure to expose a defect if one is present. For a defect to persist in the released FPGA configuration, both injection and failure to detect must occur. A high injection risk and a corresponding high detection-failure risk together produce a significant probability of a design that does not function as intended. For example, an incorrect compilation of the VHDL statements (an error in netlist synthesis) coupled with an incomplete test suite that fails to detect the injected defect would lead to a flawed configuration with respect to that defect.

In that light, the use of error-prone constructs is assigned a low-medium severity, because these constructs do not automatically lead to injection of actual defects (though they do represent poor design style). Vulnerabilities that would occur in a validation step, such as functional simulation (exposing problems that would otherwise presumably be caught during formal verification), are of lesser consequence than those that would inject defects or fail to expose defects during verification.

3.8 DETAILS CONCERNING DEFECTS INTRODUCED DURING SYNTHESIS

The following errors, listed in table 1, have the potential of introducing defects into the FPGA configuration. If these defects are not avoided during synthesis and are not exposed during testing, then they will lead to misbehavior of the configured device.

3.8.1 Poor Modular Organization

One conspicuous possible source of risk in the development of a complex PLD is unnecessary coupling of modules during the architecture step in implementation of the requirements. This issue, and the related matter of lack of locality of state variables, is well known to be important in the

context of software development. There is no easy and formulaic way to produce a design that is good from this viewpoint. For FPGA development, the same considerations arise, and the situation is made worse because hardware designers may not have everyday experience in such matters. A consequence of unnecessary coupling is the risk that fixing one problem will introduce other flaws, possibly trading an error that has been detected for a flaw that might not be caught by subsequent testing.

The probability of unnecessary coupling of modules is high, because hardware engineers do not have the training and viewpoint of software developers who will know the cost of unneeded coupling (and even in software, problems frequently arise in this area). The impact on safety is low until a fix in one problem triggers a second, less apparent problem. Other than early design reviews involving seasoned software or FPGA developers, there is no way to mitigate this issue.

3.8.2 False Routing Connections

This occurs when, because of poor tool interface design, a block diagram tool capability allows the engineer to create an apparent connection between elements of the design, which is not actually implemented in the resulting netlist. Although this is strictly speaking a user (designer) error, the underlying cause is a poorly considered tool interface. Errors of this nature have occurred in the example design flows, and in other FPGA design projects. The impact can be serious. In consequence, it is recommended that if block diagram capabilities are used, the resulting VHDL files be reviewed, checking that all required connections are reflected in the statements.

3.8.3 Mistranscription of Requirements

Another error type is mistranscription of low-level requirements into hardware statements. This occurs when the designer creates VHDL statements that inaccurately reflect the device requirements. This is purely a user error (coupled with a code review deficiency) and is normally not affected by tool qualification (or lack thereof) at the stage of entry of the VHDL statements. To some small extent this could be related to the use of tools because a bad editing interface would make it more likely that transcription errors occur. It is hard to see that tool qualification would affect this possibility.

The probability of this sort of error is high. However, safety considerations must be integrated from the start in design projects intended for aerospace applications. Therefore, errors of this nature should be caught at an early development phase by validation techniques that do make use of tools. If functional simulation is part of the design process and the test suite provides good coverage of requirements, then mistranscription of requirements will almost always be revealed before the P&R step. In that sense, tool qualification may play a role in controlling the risk represented by these errors.

3.8.4 Uninitialized Variables and Other Error-Prone Constructs

These errors can systematically be eliminated, by the use of ALINT or a similar VHDL checking tool. Because later phases of the design process will usually uncover the existence of problems of this nature, most of these are considered of low severity. However, the creation of uninitialized variables may occur frequently in the absence of systematic checking and has the potential for introducing subtle misbehavior, so this error may have a greater impact.

3.8.5 Transparent Latch Signals in Lieu of Flip-Flops

An exposition for how to avoid the problem is found in reference [8]. It is almost always a result of poor choice of VHDL statement expressions, and therefore is not introduced by a defective tool per se, although a superior tool might allow an assertion that there are no intended transparent latches, and issue an error if one is presented.

Literature review indicates that the probability of this flaw is high, although transparent latch problems did not occur at all in the two design flows studied. The consequence is potentially serious because a comprehensive test suite may not necessarily catch undefined behavior.

3.8.6 Metastable States

These generally occur when a subset of the hardware statements implies a logically inconsistent set of conditions. A tool might be configurable to consider metastable states as an error. However, the confidence that a netlist synthesizer will discover all such problems would be much lower than the confidence that the tool will faithfully and correctly create a netlist representing the VHDL statements presented.

The probability of this flaw is moderate, and it did occur at least once in one of the two design flows studied. The consequence is potentially serious, because a comprehensive test suite may not necessarily catch undefined behavior.

3.8.7 Netlist Inconsistent With Hardware Statements

This appears in table 1 as incorrect VHDL compilation, which would lead to an incorrect netlist. This is a genuine injection of a flaw by improper tool behavior. An answer to this potential source of error would be to have (and use) a netlist equivalence comparison tool to compare netlists synthesized by distinct tool suites.

The probability of this flaw is very low because netlist synthesis is a mature technique and not particularly complex or ambitious. The consequence is not too serious unless multiple flaws occur because netlist-level simulation/verification testing should reveal the existence of such flaws (although it may then be quite difficult to trace down what the problem is).

3.8.8 Partially Uninitialized Variables

This type of error relates to creating a signal that is well-defined in most cases, though the value is undefined (and the line is therefore electrically floating) for some combination of other signals and states. This type of error is almost unique to FPGA design in the hardware realm and has a software analogue of creating an uninitialized stack variable and leaving some conditional path for that variable to be used before it is assigned a value.

The probability of this sort of flaw is low, unless the module coding is poorly organized. Review of the VHDL statements will be effective at exposing such flaws. A high-quality logical-level simulation, running through high-coverage test suite, should catch this type of flaw as well, and may indicate it as a warning. In that case, it may be overlooked unless the design process demands no-warning netlist synthesis. VHDL coding standards requiring establishment of an initial value

at the same time each variable is introduced will eliminate these flaws but might in exchange leave problems in which a variable, which under most control paths is set intentionally and sensibly, is left at its arbitrary initial value under some control path. Therefore, such standards will not be a solution for this type of error.

3.8.9 Timing Rules Inadequate to Ensure Proper Performance

This is an injection of a flaw due to an error on the part of the developer. Often, the case is failure to include timings for reset or subsidiary clock signals in the VHDL timing file. In other cases, the developer may be making incorrect assumptions about the timing of signals on the circuit board. These flaws appear in table 1 as incorrect or incomplete timing constraints.

The probability of this kind of flaw is moderate; it was seen in multiple instances in both of the studied design flows, and mentioned by other developers. The consequence of omitted timings is not serious, and simulation tools can catch these. The consequence of not understanding the interface to the circuit board is much more serious, because arbitrarily subtle misbehaviors may be introduced. However, this is generally the case in hardware design, and as long as the developer propagates any process that protects against such mistakes in general, to the specific case of providing these timings, the probability of a flaw with serious consequence becomes very low.

3.8.10 Multiple Time Domain Problems

As discussed in section 3.1.1, FPGA configurations that involve several time domains that are not synchronized present subtle issues and opportunities for injection of defects that may be difficult to expose. Although the example design flows did not suffer from defects injected in this way, several other projects discussed as part of this investigation did encounter such defects.

Injection of these problems is not an issue that can be addressed by synthesis tool qualification. However, subsequent verification activities, which will be performed relying on verification tools, are critical in detection problems of this nature. In that sense, proper operation of those tools becomes critical to safety assurance, and this raises the issue of qualification of the verification tools.

3.8.11 P&R Fail to Follow Timing and Signal Rules

This is a genuine injection of a flaw by improper tool behavior. It appears in table 1 as erroneous timing rules (the P&R tool has been supplied a set of timing rule data that does not reflect the reality for the FPGA device) or incorrect application of (correctly provided) timing rules.

The probability of this flaw specifically due to tool malfunction is low, though higher than that for synthesis being unfaithful to the netlist logic, because timing issues are always more subtle. The provider of this tool is dealing with dozens of chip types, each of which needs proper routing tools, and this can increase the probability that some of these tools behave imperfectly. The consequence is serious, because a problem here can introduce timing-dependent misbehavior. This may be an area that would greatly benefit from some form of tool qualification.

3.8.12 P&R Not Faithful to Netlist Logic

If the synthesis of the gate and connectivity configuration for the actual hardware does not accurately represent the logic implemented by the netlist, this is a genuine injection of a flaw by improper behavior of the P&R tool. Unlike the case for improper netlist generation, improper P&R cannot be controlled by comparison of the outputs of distinct tools, because the routing is non-trivial, the algorithm is proprietary, and optimization is performed. This is reflected in table 1 as "Layout does not reflect netlist."

The probability of this flaw being specifically due to tool malfunction is low, though not negligible. The consequence is not always serious because an outright logic change will normally be exposed during testing. However, in rare cases, the flaw might lead to a metastable state, which has more potential for uncaught misbehavior.

3.8.13 Improper Configuration of Device From Configuration File

Another way in which the FPGA configuration could fail to reflect the netlist is for an error to occur in the step of configuring the FPGA based on an otherwise proper file generated by the placement step. This is purely a hardware error and is mentioned in this report only because it is almost completely specific to development of FPGAs and other complex PLDs.

The probability of this sort of flaw is very low. Experience spanning quite a few recent FPGA development projects revealed no instances of improper translation of the configuration file into the device configuration. In addition, if full-coverage hardware testing is performed, this sort of flaw (if present) would be revealed.

3.9 DEFECTS ASSOCIATED WITH VALIDATION AND VERIFICATION

High levels of assurance come about if a device is designed without injected flaws, and a set of verification activities is applied that would expose any improper behavior that exists. Therefore, it takes a combination of two very unlikely occurrences for a defect to remain in the configured device.

Errors and risks of injecting defects were discussed in section 3.1.7. The following errors can potentially cause the verification activities to fail to detect a behavioral defect, if any defects have been injected during synthesis activities. With verification errors, this list includes validation errors, which may prevent early detection of defects.

3.9.1 Incorrect Functional Logical Simulation

This will result in a validation step not catching existing flaws, or much more likely, exposing problems that are not real.

The probability of this flaw is low, and the consequence is minimal, in that validation steps are performed so as to expose flaws at an earlier stage when they are easier to rectify and not directly relied on for safety assurance. Rare errors in validation are of little consequence. Frequent errors of this sort will cause the developer to abandon the simulation step, losing whatever safety benefits and development efficiency improvements that step was providing.

3.9.2 Erroneous Timing Rules in Simulation

The step of validation by detailed simulation of the chip configuration can expose defects that might not be revealed during verification testing on one or two specific devices. Such defects involve missed timings, which are out of specification per the device timing rules but are close enough that the chip used during verification nonetheless behaves properly.

This validation can be valuable, particularly if the designer has some idea of where problems are most likely to occur and can focus the validation effort on those areas. However, if the simulation tool applies improper timing rules, then this validation will be much less valuable.

A related possibility is that of incorrect operation of the simulation tool itself. Although no such flaws were seen in the course of investigating for this report, tool qualification is potentially relevant to this issue.

3.9.3 Inadequate or Inaccurate Timing Checks

Detailed timing checks occur in the course of validation of the P&R. These can be performed by simulation or by signal tap techniques using a configured FPGA. The goal is to ensure that signals settle at their proper values at times that are early enough to meet the device's timing rules. These checks are among the most valuable techniques in assuring that successful completion of requirements-based testing will translate to consistent proper operation of every device in the field.

The timing validation phase can be suboptimal if the designer's timing tests fail to cover critical timings. The risk of this is high, because the number of potential timing observations is very large, and only a small fraction of these are critical (in the sense of being at risk for missing the nominal device timing rules). The principal technique for minimizing this risk is a review of the plan for detailed timing tests.
A similar potential error, which is of much lower risk, is that of improperly interpreting the results of the timing checks.

The timing validation phase can also provide misleading information if the timing measurement tool malfunctions. Whether the tests employ a tool that performs detailed timing simulation or a tool that taps into on-chip signals and feeds the data to virtual oscilloscopes, timing validation will rely on a sophisticated tool. Because the detailed timing checks are a validation step, as opposed to a verification that will be submitted for certification credit, safety assurance does not in principle demand qualification for these tools. Nevertheless, it is felt that it would be beneficial to encourage activities aimed at increasing the confidence that the results of timing studies are accurate.

3.9.4 Incomplete Verification Against All Requirements

If verification is incomplete, in the sense that some requirements are not tested, then DO-254-based safety assurance relying on that incomplete verification will not be sufficient to substantiate certification credit. This is particularly relevant for FPGA development because, in the case of verification of FPGA's, particularly in-situ on the actual circuit board, it has been noted that injecting the full suite of test vectors into the verification test may be challenging.

The probability of incomplete testing is very high, unless conscious steps are taken to deal with this issue. The consequence may be negligible or serious, depending on whether any actual flaws were overlooked. In a safety assurance context, because incomplete testing leaves no way to demonstrate the absence of actual flaws, the consequence must be considered serious. In cases in which there was a testing review of the test bench used in simulation, and in which the test bench is not automatically carried over to the hardware verification tests, the PHAC should require a second review of the coverage provided by the hardware tests.

This issue is reflected in table 1 as "Incorrect supplying of test vectors." As noted in that table, if this step is automated, that can reduce the likelihood of failing to apply some intended test vectors.

3.9.5 Inaccurate Test Environment

Properly structured suites of verification tests have the ability to reveal both repeatable and sporadic defects in functionality. However, if the environment applicable when those tests are run is materially different than the intended operating environment for the FPGA device, then the information revealed by these tests will be much less valuable. In particular, low-level sporadic misbehaviors might be missed because the test environment lacks some unfavorable aspects of the actual operating environment, which might increase the likelihood of those misbehaviors.

Tool qualification may play a role in reducing the likelihood of such problems, because tools (such as CTS) providing a custom board environment and a means of applying test vectors are becoming a valuable asset in verification of FPGAs.

4. SAFETY ASSURANCE CONSIDERATIONS FOR COMPLEX PLDS

This section presents some aspects of the safety assurance activities performed when designing leading-edge FPGA components for avionics applications. The guidance used for FAA certification will be summarized only, because the target readership of this report is expected to be deeply familiar with the DO-254 guideline document and the FAA certification process. However, current practice regarding qualification of FPGA design tool suites will be discussed.

4.1 CHAIN OF ACCEPTANCE OF COMPLIANCE TECHNIQUES

This section presents the chain of documents that define acceptable techniques for demonstrating compliance with the relevant FAA regulations.

For commercial air transport, the driver of the FAA certification process is Title 14 of the Code of Federal Regulation (CFR), Part 25, Section 1309, which requires, among other considerations, that:

Item (b1): The occurrence of any failure condition which would prevent the continued safe flight and landing of the airplane is extremely improbable.

Paragraph (d): Compliance with the requirements of paragraph (b) of this section must be shown by analysis, and where necessary, by appropriate ground, flight, or simulator tests.

For systems intended for use in air transport aircraft, Advisory Circular 25.1309-1B describes acceptable means of showing compliance with these paragraphs of the regulation. In particular, AC 20-115C, DO-178C, ARP4754A, and ARP4761 are referenced as acceptable techniques for demonstrating compliance. The Aerospace Recommended Practices documents (ARP) are considerations and guidelines about general safety assessment methodology, but ARP4761 specifies DO-178 and DO-254 as the companion process documents for software and hardware, respectively.

Whereas AC 20.115C recognizes software process documents acceptable for showing compliance with applicable airworthiness regulations, AC 20.152 explicitly recognizes DO-254 as an acceptable means of compliance for electronic hardware aspects of type certification. The FAA has issued FAA Order 8110.105, whose subject is "Simple and Complex Electronic Hardware Approval Guidance." The software analogue of this is Order 8110.49, Software Approval Guidelines. In particular, Order 8110.105 specifies that DO-254 may be used as a means of showing compliance, and significantly clarifies interpretations of sections of DO-254. FPGAs are considered electronic hardware, implying that DO-254 applies.

Overall, the picture of "which document allows acceptance of which techniques" is given in figure 2. An immediate generic observation is that although the design of a complex PLD has much in common with software development, because the component is considered to be complex hardware, no document in the guideline documentation tree points to, accepts, or even suggests the use of DO-178C (Software Considerations) in showing compliance for these designs. It is likely that safety assurance for complex configurable electronic hardware would benefit from involvement of at least some of the guidelines in DO-178C that are not present in DO-254. Because these software-related considerations are being studied in the context of other FAA efforts, they will not be discussed at length in this report.

Figure 2 shows the chain of acceptance of assurance techniques in block form. DO-178C recognizes a technology supplement DO-330 "Software Tool Qualification Considerations" [4]. Note that for electronic hardware falling under the control of FAA Order 8110.105, figure 2 shows no explicit path justifying, for example, DO-330 as a means of demonstrating safety assurance for the design tools used.

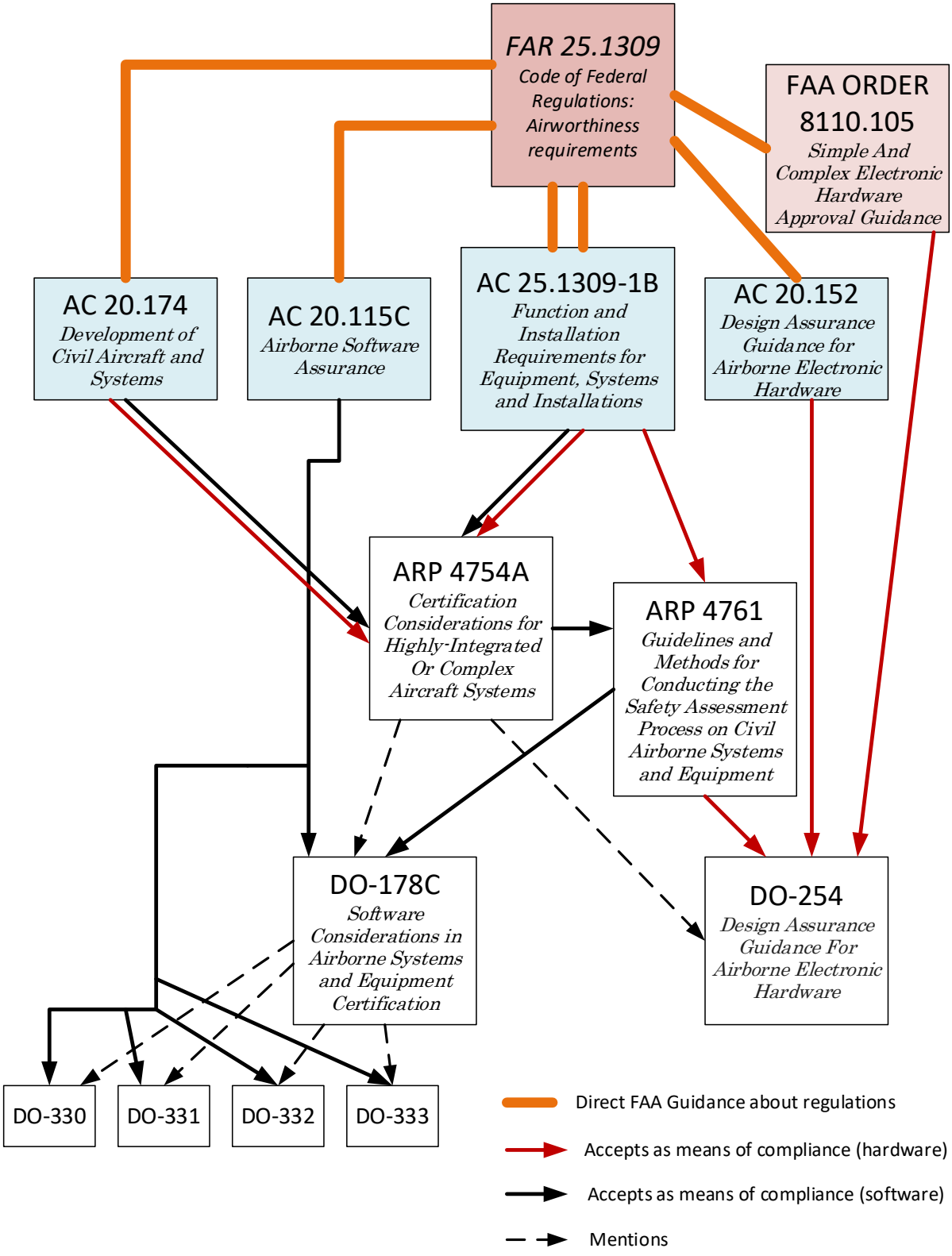

Figure 2. Chain of acceptance of compliance techniques

Some subset of the DO-330 considerations might be applicable to tools used in FPGA design, but DO-254 has an earlier publication date and does not recognize DO-330 or any alternative analogous document. Future modifications to DO-254 may address this issue. Currently, there is a need for some clear statement that the same tool qualification considerations should be applied to FPGA development and verification tools, perhaps with restrictions and/or changes recognizing the distinctive nature of FPGA development.

The job aid for Airborne Electronic Hardware Review simply indicates the review of tool qualification plans and the evaluation of the rationale for why a tool need (or not) be qualified. No recommendation beyond section 11.4 of DO-254 is indicated in the job aid.

The clarifications in DO-248C address tools in both the FAQ and the Discussion Paper sections. However, the focus is on formulating criteria for deciding whether a tool should be qualified. Besides the fact that DO-248C content is not normative, it does not provide clarification for the problem at hand. At any rate, DO-248C, consisting of supporting information for DO178C and DO-278C, is a software document, so the same issues about applying it to FPGA development may be raised.

4.2 SAFETY ASSURANCE FOR SYSTEMS INCLUDING PLDS

The design of gate configurations for FPGAs and other complex PLDs has much in common with software development, and safety assurance considerations akin to those applicable to software development are logically applicable to FPGA development. However, a broader spectrum of sophisticated design tools play an essential role in FPGA development than may be the case for classical software development. The essential FPGA design tools encompass such a wide range of capabilities that general qualification of a given tool may be impractical given current guidelines and recommendations.

The usual approach to providing safety assurance, when a tool that could not itself be qualified has been used, is to provide independent verification activities. When an FPGA development tool plays a role analogous to that of a software development tool, the term “qualified tool” is used in the same sense as it is used with respect to DO-178C software certification. For example, in the software realm, compilers generally are not explicitly qualified and the artifacts that would make them qualified are not available. This is dealt with by providing comprehensive independent tests, verifying that the code generated by applying the compiler (tool) to the source code has correctly implemented the requirements. As a function of development assurance level, additional activities may be applied to demonstrate that the safety objectives have been met. For example, for software intended to perform DAL A functions, explicit examination of the output of the compiler is also used.

In the case of a complex PLD, manual design of the device configuration is much more error-prone than relying on the available generation tools. However, a comprehensive manual review of the generated configuration may, in many cases, not be feasible. Therefore, device and system assurance will need to rely on the comprehensive verification route.

When a complex PLD is designed with the aid of tools that cannot be qualified, with the intent of certifying the resulting configured device by comprehensive verification, two issues may arise.

The first issue is that the best available verification tools may be provided by the same vendor who provides the design tools, leading to the possibility that a single misconception causing an error in generation can also cause acceptance of that error in verification. This may not be a problem in principle if the configuration is verified by subsequent steps; that is what allows the use of such tools in the DO-178C context. However, this brings up the second issue: As the complexity of the PLD increases, "comprehensive" verification may become exponentially more difficult to achieve and be overly time-consuming and cost-prohibitive. For example, inputs that are unused or ignored under some set of circumstances still have the potential to affect the output of the device; including all states of such inputs in the comprehensive verification can create a combinatorial explosion of cases to verify.

4.3 ELEMENTAL ANALYSIS CONCEPTS WITH IMPLICATIONS FOR FPGA DEVELOPMENT

Some areas in which DO-254 is less "heavy" than the analogous software guidelines in DO-178C—in the sense that a process adhering to all guidelines and steps recommended by DO-254 would omit some activities specified by DO-178C—involve elemental analysis. In software, this includes the concepts of code coverage, decision coverage, and elimination of dead code. As already noted, per DO-254, tools used exclusively to verify coverage in an elemental analysis need not be qualified beyond a claim for credit of relevant tool history. This may place a limit on the burden involved in tool qualification, but it also indicates that tool qualification concerns are relevant in these areas.

4.3.1 Code Coverage

Software assurance can involve demonstrating that each code statement can be traced to the low-level requirement driving the need for that statement (which in turn is traceable to the high-level requirements). In the case of FPGA development, this can be translated as: each VHDL statement that drives the netlist creation is traced back to the low-level requirement that necessitates that HDL statement.

For software assurance, DO-178C-based safety assurance includes demonstrating that every line of code has been exercised during the set of verification tests. The analogue for an FPGA design would be to show that every gate specified by the netlist is "used" in the sense that if it were replaced by a simple pass-through, at least one test would fail. PLD development tools facilitating this form of elemental analysis are beginning to emerge. The issue of tool qualification will be important.

4.3.2 Decision Coverage

Assurance guidelines for software intended to perform DAL-B (or DAL-A) functions specify decision condition coverage. For each conditional, both (or all) possible outcomes proceeding from that branch point must be demonstrated for the testing exercises. In PLD design, the analogue of a conditional is a gate implementing a branch. A very high fraction of the gates are implementing branch points of one sort or another. This branch-coverage analysis would be difficult to demonstrate in an automated way, unless qualified coverage tools are available.

DO-178C software assurance guidelines specify modified condition decision coverage (MCDC) for modules intended to perform DAL-A functions. The analogue of this for PLDs is to specify that that every allowable input state of each gate, or local group of gates, should be encountered in the course of the full suite of tests. In FPGA development, this is called focused expression coverage (FEC). The intent is to create stimulus that is so complete it would catch a synthesis logic error by fully simulating every "interesting" set of inputs. This is manifestly impractical without tools to automate the creation of these inputs and test the combinations in an auditable way. Such tools exist, and to the extent that FEC is relied on as a component of verification, the correct performance of these tools may be at issue.

4.4 ELIMINATION OF DEAD CODE AND UNEXERCISED CAPABILITIES

For ordinary software, elimination of dead code (code that will not be executed during operation of the module) is a logical consequence of the statement coverage guidance. However, for FPGA development, there are considerations that might (depending on interpretation) make the concept much more onerous. This occurs if the guideline is taken to mean that every capability that is implemented by configuration of the gates on this device must be used, and its usage must be traceable to the satisfaction of some requirement. This is a rule of “no implemented but unused capabilities.”

In the context of FPGA development and assurance, the “no implemented but unused capabilities” stricture raises two serious possible issues. The first issue is that one often “drops” a packaged set of gates developed by a third party (i.e., an IP module) into the FPGA for use in handling a complicated chore, but that IP module likely contains additional capabilities that are not used in the context of the FPGA functionality. The second issue arises by virtue of the fact that the selected FPGA device will likely contain internal “hard” sections implementing sophisticated capabilities. For example, the Cyclone 5X FPGA has two built-in ARM9 processor cores. An application may well select this device because of its speed, robustness, I/O capabilities, and/or number of available gates, and yet make no use of these dual processors

The matter of unused capabilities when IP modules are used in PLD development is being addressed by other efforts and will not be addressed in this report. However, this report does discuss issues of safety assurance regarding the use of unqualified IP modules, which may potentially lead to flawed performance relating to their intended functions.

4.4.1 The Issue of Unexploited Device Capabilities

Unexploited device capabilities, which also might be considered as the hardware/configuration analogue of unused code paths, are a related issue, clouding the process of device verification. The developer usually cannot excise all capabilities other than those exploited to meet the requirements. Often, submodules on an FPGA (such as a network interface block) will have internal configuration, which the design at hand uses to tailor them to the requirements (e.g., an ARINC 429 communications link as opposed to an Ethernet or MIL-STD-1553 link). The other possible configurations might never be realized in the design created (e.g., a design intended for certification for commercial transport aircraft would not involve MIL-STD-1553 links). Verifying that only the capabilities required to meet the requirements are activated, under all input and internal state combinations, can be challenging.

The proper consideration for such capabilities is that they need not be certified or verified if they are not used, but at high DALs, acceptable evidence of partitioning (i.e., evidence that the presence of the capability cannot affect the performance of the verified functionality) should be required for such unexploited capabilities.

4.5 MULTIPLE DALS ON A SINGLE FPGA

The question arises whether a single FPGA can be used for a high-DAL function and also a low-DAL function within the same system. DERs have given presentations indicating that this is possible, but concerns remain. The key question is whether partitioning and isolating one function from the other can be demonstrated at a sufficient assurance level.

One useful technique for demonstrating assurance touches heavily on some tool features. Assume place and route can be performed in a reproducible manner. If the tool that does this also allows the lockdown of some set of gates and signals, then the following technique provides significant evidence that the high-DAL functions are unaffected by the presence of the low-DAL functionality (and therefore that partitioning is demonstrated):

- Design the high-DAL functionality, and create a configuration that passes all verification tests for these capabilities.
- Run the full verification suite covering all requirements for the high-DAL sector on the hardware in that configuration, and capture the timings and the results. This requires tools to automate supplying the test vectors and capturing the timings.
- Repeat this run and compare timings to quantify the inherent level of timing fluctuation.
- Implement and functionally simulate the low-DAL requirements.
- Create a configuration in which the logic elements and connections from the high-DAL configuration are locked down in places, and the netlist for the low-DAL sector is used as the input for further P&R.

    - This requires a place and route tool supporting the lockdown feature.
    - Design engineers using Xilinx tools state there is a way to lock the design.

- Run the verification suite covering the low-DAL requirements on the hardware in that configuration to verify that that sector works.
- Run the full verification suite covering all requirements for the high-DAL sector on the hardware in that configuration, and capture the timings and the results. Verify that the configuration still passes all high-DAL verification tests.
- Compare the timings of the two high-DAL verifications. If the configuration has passed all high-DAL verification tests, and the timings are identical (modulo the inherent level timing fluctuation), then that is strong evidence that the presence of the low-DAL sector has not affected the high-DAL sector.
- Finally, all the external input lines, including the low-DAL inputs, must be treated as if they were high DAL. This deals with the concern that no matter how well separated the functions are, if in the course of using the low-DAL sector you place a connection to a 28V input, it will adversely affect the high-DAL sector by destroying the FPGA.

Passing all high-DAL verification tests, with no out-of-range timing discrepancies, constitutes proof of the effectiveness of partitioning, provided that the set of high-DAL verification tests is complete with respect to all normal and robust-case high-DAL requirements.

This report recommends that at least this level of evidence of isolation should be expected when asserting partitioning as a component of demonstrating safety assurance of a multi-DAL FPGA.

4.6 GAPS IN CURRENT ACCEPTED MEANS OF COMPLIANCE

A gap in the set of accepted means of compliance can be an instance of a solid and useful assurance technique that is not accepted or mentioned in any of the FAA documents or the chain of documents they accept as means of demonstrating compliance. Because of the nature of device complexity, which has greatly increased in recent years, there arise possibilities of error risk that might survive undetected despite adherence to the recommended assurance techniques. An example of the latter is that prudent FPGA design should involve a review of timing constraints and reset completeness; whereas any individual DER might suggest such a review, the set of guidelines lack mention of that activity.

One area that may benefit from additional guidance is the use of elemental analysis. Elemental analysis is discussed in appendix B of DO-254 as an “advanced” method that may be used in satisfying the objectives of the verification process specified in section 6.2 of that document. That appendix describes how elemental analysis may be used to show that functional failure paths are verified by associated verification test cases. However, guidelines broadening the scope of application of elemental analysis could be presented. Because configuration development for a complex PLD has characteristics that are very similar to those of software development, elemental analysis techniques analogous to software statement and/or decision coverage (per DO-178C) can be an important aspect of complex PLD safety assurance.

Given that elemental analysis can be an important component in approval of FPGA designs, it would be beneficial to provide guidelines and recommendations to encourage uniform application of these techniques in the approval process. In the cases of FPGA design, the analogue of statement coverage is verification that each gate or block of logic elements has been exercised in the course of a suite of tests. FEC, the analogue of decision coverage, adds verification that each valid combination of inputs to each element has been exercised. Both of these types of verification are close to impossible without some well-crafted tools to automate the evaluation, and fresh development of such tools as part of each FPGA design process is impractical. Therefore, elemental analysis will rely on a set of simulation and analysis tools. Per the statement in section 6.3.2 of DO-254, these results alone “cannot be used for the purpose of certification credit without supporting evidence of their validity.” Because the elemental analysis tools used will be common to many PLD design efforts, this brings to the forefront the issue of how such tools can be qualified to ensure a high probability of correct tool behavior and to allow their results to be creditable in safety assurance assessments.

An important category of potential errors are those that relate to the hardware/timing nature of the FPGA devices. Although in designing configurations for complex PLD configurations, most design flaws that lead to safety related errors would be exposed by functional testing techniques analogous to those used for validating and verifying software, these techniques may be inadequate

to expose potential timing-related errors. To achieve assurance of proper performance, further justification demonstrating that such flaws are of sufficiently low probability (as appropriate for the DAL associated with the intended function of the device) should also be presented. Current guidelines may not offer sufficient assurance techniques in this area. To the extent that vendor-supplied tools are used in demonstrating proper timing, the issue of assurance of correct tool behavior again becomes relevant.

4.7 AIRBORNE FPGA DESIGN ASSURANCE IN NON-FAA CONTEXTS

This section examines safety assurance treatment of FPGA development in non-FAA contexts that are related to airborne applications. EASA certification guidance and U. S. Air Force procedures for demonstrating safety were examined to uncover potential considerations that could beneficially be applied to the set of FAA guidelines.

Although these organizations share with the FAA a high degree of concern for safety assurance, their examination uncovered only one minor addition to existing FAA guidelines in areas relevant to tool qualification and FPGA development.

4.7.1 EASA Certification Memoranda

The relevant document [10] issued by EASA is CM-SWCEH-001, "Development Assurance of Airborne Electronic Hardware." This certification memorandum, issued in 2011, provides guidance material on certification aspects associated with the use of airborne electronic hardware including PLDs. Sections 3, 4, and 8 of this document harmonize with and address the same topics as FAA order 8110.105 [11].

Within these sections of the EASA memorandum, the only content that is relevant to the tool qualification concerns of this research, yet is not present in FAA order 8110.105, is:

- An explicit mention of the fact that, per DO-254, tools used exclusively to verify coverage in an elemental analysis need not be qualified beyond a claim for credit of relevant tool history.

4.7.2 FPGA Design Assurance in the Military Aircraft Context

Military aircraft context refers to the U.S. Air Force process of certification and authorization of devices and systems for use in aircraft. Naval aeronautics and Army aviation have their own processes, which are similar to the Air Force process.

In the military aircraft context, there are three classes of roles for devices and systems (referred to as "items"), leading to three logical levels of design assurance requirements and three levels of stringency for certification of original design and later modifications. The classes are:

1. Flight Critical (sometimes called Flight Safety Critical)
2. Mission Critical
3. Other

Development and modifications of items are also categorized in terms of their role, as being:

- Reportable: All Flight Safety Critical items are reportable, and most Mission Critical items will be designated as reportable. Initial development and all modifications of reportable items go up to the Airworthiness Authority (AWA—see immediately below).
- Non-reportable with airworthiness impact.
- Non-reportable with no airworthiness impact.

The combination of criticality, reportability, and airworthiness impact plays a role similar to that of the DAL in determining the appropriate nature of safety assurance activities. There is no documented matrix directly translating each combination of criticality, reportability, and airworthiness impact to a specific equivalent DAL.

There are three levels of certification authority. The most stringent level is review by the AWA. This is a single designated high-ranking officer responsible for approval of all reportable items and modifications. This approval is called an Airworthiness Certification Action.

The airworthiness certification criteria are driven by Military Handbook 516-C (MIL-HDBK-516C) and expressed as questions listed in the Airworthiness Directory. Whereas these include many questions relevant to PLD design, in every instance the accepted means of demonstrating answers to such a question goes through the DO-254 process. Therefore, careful perusal of the design assurance process in the military aircraft context reveals no new insights into tool qualification or assurance of PLDs.

5. FPGA DEVELOPMENT TOOL QUALFICATION CONSIDERATIONS.

The primary objectives of this report are to recommend appropriate means of safety assurance in cases in which FPGA development relies on tools for which qualification by an FAA process would not (at present) be straightforward. There might be several reasons why this would be the case, including:

- Lack of access to tool-creation artifacts.
- Lack of evidence of thorough testing of the tool.
- No FAA defined process of qualification applies to the tool and context of use.

If a component used in a design suffered from an inadequate component design process or lack of artifacts to demonstrate safety assurance, then it would be difficult or impossible to incorporate that component and still satisfy safety assurance guidelines for DAL-C or higher. The situation regarding tools that assist the development in the course of creating the component, validating the design, verifying proper performance, or assessing the extent of testing coverage is more subtle. For example, section 6.3.2 of DO-254 states that “simulation results alone cannot be used for the purpose of certification credit without supporting evidence of their validity.” In the case of results obtained by use of a tool or set of tools, that supporting evidence would need to address two areas:

- Was the tool used correctly to create a meaningful and comprehensive set of simulations?
- Is the tool sufficiently reliable that the correct usage will lead to trustworthy results?

Because FPGA design tools are generally not created from scratch by the component developer, the tool reliability concern brings the issues of tool qualification and use of unproven design tools to the forefront.

5.1 AREAS IN WHICH TOOL QUALIFICATION WOULD BE BENEFICIAL

Section 11.4 of DO-254 presents tool qualification considerations. The tool assessment concepts in section 11.4.1 are just as appropriate for FPGA development as for other tool usage. However, as noted in paragraph 9 of that section, strategies appropriate for qualification of other software tools "or other means acceptable to the certification authority" are necessary for devices performing DAL-B or DAL-A functions. This may be considered to bring in DO-178C as a resource for assurance of FPGA design tools. These strategies may be overly stringent for devices intended to perform DAL-C functions, but some form of similar statement, perhaps specifying certain aspects of DO-178C as being applicable to FPGA development, would be helpful.

In addition, appendix B of DO-254 presents considerations applicable to the use of libraries of IP blocks. To the extent that these are considered as FPGA development tools, design assurance methods described in section 3 of appendix B may be considered as relevant to tool qualification for use at DAL-B and DAL-A.

There is already a tool qualification standard—DO-330—that is applicable in the very similar realm of software development. This report recommends applying all or part of this standard to FPGA configuration development tools.

Most mature netlist synthesis tools could potentially be qualified (under some reasonable definition of qualification). This would control the risk of injecting defects by using a tool in an unintended operating environment or using a flawed synthesis tool. This report recommends guidelines facilitating a qualification mechanism that the major and mature netlist synthesis tools can meet.

Place and route tools have high safety-assurance risks, and it would be desirable that early "shake-out" of possible errors would occur in less safety-critical contexts than FAA-certified aircraft. It may be wise to have some registry of combinations of chip versions and P&R tool releases that have accumulated sufficient service history to be considered to have suitable reliability. This report recommends guidance that would help make possible some form of registry of combinations of chip versions and P&R tool releases that have accumulated sufficient service history to be considered to have suitable reliability.

Because the usual issue of essential changes associated with each move to a different FPGA generation will make qualification of detailed timing simulators difficult, the authors argue that tool qualification is not feasible in this area. This report recommends accepting results of timing simulations as verification evidence only if a sample of those results are cross-verified by examination of actual hardware timings.

Tools to automate parts of the verification process can be qualified to increase the degree of trust that the full supplied suite of test vectors was properly applied, and that the results were properly compared to the expected results. Because software vendors are already aware of the benefits of making these tools suitable for development of devices performing functions at DAL-C and higher,

it is reasonable to provide a clear path toward some form of qualification of such tools. This report recommends guidelines to that effect.

5.2 PROSPECTS FOR QUALIFICATION TOOLS BEING USED IN FPGA DESIGN

Although the primary scope of this research deals with issues related to the use of unqualified tools, this section briefly describes the situation concerning qualification of existing FPGA development tools. Whereas the use of any tool in a specific design may be assessed for safety assurance by independent assessment and verification, tool qualification adds information that may be used to lighten that burden in multiple and unrelated designs.

These development tools tend to be packaged into all-encompassing tool suites or development environments. When considering qualification of functionality that is implemented and packaged in that way, it may be nearly impossible to qualify the entire suite, and there may be many parts of the suite that do not need to be qualified. A possible concern in that situation is that it might be difficult to separate out the tool that needs to be qualified and not deal with the whole suite. For FPGA development tools, at least those offered by the major FPGA vendors Altera and Xilinx, and those offered by software development vendors Altera and Mentor Graphics, this worry can be laid to rest. The tools suites are partitioned into tools that may be purchased separately, which implies that the functioning of each tool is independent of the presence of other tools.

The synthesis tool chain creates the FPGA configuration. Two crucial synthesis steps are netlist synthesis (starting from designer-supplied HDL functional statements) and placement/routing, which creates the configuration from the netlist. An important class of potential errors in FPGA development is related to improper behavior in the chain of synthesis tools used to design the FPGA. This can include incorrect netlist synthesis from the HDL statements, grossly improper logic element placement, and subtle straying from the P&R constraints applicable to the timing intended for the device.

Whereas the creation and verification activity types are of equal importance from the safety assurance standpoint, the nature of qualification requirements for tools might differ. Any flaw in a tool used in the creation chain can directly lead to malperformance of the device. Flaws in a tool used for validation or verification may lead to false reports of defects, or reduced probability of detection of any device performance flaws that might exist, and should not themselves introduce performance flaws. This is a fundamental difference and might affect tool qualification issues. Because there is no choice but to use the PLD design tool chain, the issue of qualifying these tools comes to the forefront.

Assurance of the synthesis tool chain by independent assessment is awkward because two different toolsets can produce markedly different (yet both valid) logic element P&R. Independent assessment is unambiguously defined only when there exists a unique “proper” output or behavior, so that differences between the item being assessed and the comparison item indicate errors in one or the other (or both). The issue of demonstrating compliance by independent assessment in cases for which there can be ambiguities and multiple correct outputs is subtler.

However, demonstrating tool qualification based on artifacts of the tool design may be awkward or impossible, even for extremely well-designed tools. Vendors may or may not possess internal

documentation that would support the demonstration of compliance with the applicable safety objectives, and if these documents exist, they likely will either want significant payment for access, or once-and-for-all qualification by an authority agreeing to some form of non-disclosure.

A third avenue for tool qualification is relevant product experience and software service history. Demonstrating tool qualification by presenting service history can be effective for tools sold by a vendor who:

- Actively solicits feedback, especially negative feedback, from users
- Responds to error reports by providing fixes
- Makes public the history of feedback and response
- Makes public a wide selection of developments using the tools
- Has an adequate body of service history

Information about most tools' usage will be deficient in one or more of these considerations, which again leads to a difficult situation regarding tool qualification.

5.2.1 Circumstances Impeding Tool Qualification

Whereas the major FPGA software tool vendors, in particular, are moving toward making it possible to qualify their tools, there are several circumstances currently impeding tool qualification.

The first difficulty is obtaining information that is adequate for qualification based on product experience. There are two sides to this problem: reticence on the part of the vendor to make public open and unresolved reported issues and lack of publicly available records of successful use. The latter can be overcome within a single large company if sufficient projects within that one company have successfully used a specific tool.

The issue of public problem reporting is more difficult, as the standard corporate mindset is to avoid admitting to any defect until it is "established in a court of law." However, these vendors are anxious to cooperate and make their tools suitable for use in applications requiring high levels of design assurance. A clear set of guidelines specifying what form of history evidence is suitable for tool qualification is recommended. Vendors do actively collect errata and currently tend to make them publicly available when a fix is available. Moving that availability up to the time when the existence of the problem is verified would facilitate experience-based qualification; this report recommends guidelines encouraging early publication of reported deficiencies.

The second difficulty is the lack of visibility of the tools' implementation and life-cycle data. The data, necessary for design-process-based tool qualification, are not routinely made available and may not exist at a level suitable for application of DO-178C to that software. This approach to qualification of FPGA design tools will generally be inapplicable.

The third difficulty is the fact that FPGA design tools inherently have to deal with a wide and rapidly evolving variety of target devices, and are used on a variety of operating systems in different environments. If tool qualification is taken to demand that the version of the tool being qualified must be frozen, and only that specific version is credited with qualification, then

designers will not be able to use qualified tools except when they are configuring some specific (assumedly older) FPGA. This report recommends that some form of qualification of the "kernels" of these tools ought to exist, and that there be some mechanism to independently assess just the variations that fall outside this qualification.

The issue of usage on a variety of operating systems is a real challenge. It is reasonable to place the burden on the FPGA developers to use a system for which proper tool behavior has been demonstrated.

5.2.2 Vendor Activities Affecting Qualification Prospects

Given that both the FPGA vendors and the independent FPGA development software vendors make mention of DO-254 and DAL levels in descriptive product information, it is reasonable to assess activities on the parts of these vendors that might contribute to or hinder tool qualification. In some cases, vendors provide tool qualification data packages for a cost, including tool functional requirements and test cases. However, that does not necessarily include all the artifacts needed to qualify the tool per DO-254/DO-330.

One area of deficiency is in tool documentation. Not all the features of these tools are documented; several projects discovered useful or necessary functionality either by trial and error or by conversation with the vendors' technical support engineers. This is only a slight deficiency. The FPGA designers generally found that, in combination with the option of contacting support engineers, the vendor documentation contained information that was adequate to make use of all features that were necessary for performing the FPGA designs. A mechanism for user-experience-based improvement of the tool documentation would be beneficial.

Another area of concern is tracking, publicizing, and repairing discovered tool defects. An example is the reaction to a bug discovered in an Altera IO block. There is a mechanism for users to report defects, and Altera has a FogBugz internal ticket system for reported defects. The observation is that most bugs were in the FogBugz system, but this is a non-public internal system. (On the positive side, the consulting technical support staff did not try to hide the contents of the ticket system.) Three months after this specific bug was reported, the fix was made available.

Similar comments were made about the release of the Vivado® generation of Xilinx FPGAs. Engineers consulted for this research indicated that this release contained a substantial number of defects when it first came out; contributing to the problems was the fact that a new interactive development interface—Vivado Design Suite—was issued, replacing the ISE® Design Suite that had been in place for many years. There is no backward compatibility on this advance; the older interface does not support the Vivado generation of FPGA devices introduced in 2012. New software is released every 6 months or so.

The impression among FPGA designers using Xilinx and Altera FPGAs is that if an early release is needed for a bug fix, the vendor will cooperate only for developers whose prospective purchase volume is extremely large; avionics and aerospace applications do not have the volume to get this type of responsiveness. The cycle of frequent releases raises complications for tool qualification in a DO-330 context. Qualification tends to apply to one version of the tool; however, precluding

the use of more recent versions that incorporate repairs of defects is counterproductive from the safety assurance viewpoint.

5.3 QUALIFICATION OF NETLIST SYNTHESIS TOOLS

The prospects for qualifying netlist synthesis tools are reasonable. These tools are mature and do not need to change when a new target FPGA becomes available.

5.3.1 The DO-254 Approach to Qualification of Netlist Synthesis Tools

It has been the experience of FPGA designers, at least over the last decade, that the compilation of the netlist from VHDL statements by the netlist synthesis tools rarely or never introduce errors. This raises the possibility that some subset of these tools can be qualified, such that their outputs can be presented as being trusted to be correct. Vendors have begun to make qualification artifacts available (at a cost).

This section assumes a tool would be qualified according to the guidelines in section 11.4 of DO-254. There might be justification for considering these tools to be so closely akin to software development tools that directly applying DO-330 makes sense; this discussion is restricted to DO-254 guidance. Only the possibility of qualification based primarily on service history is considered, as opposed to the strategy described in appendix B of DO-254, which is oriented more toward an analytic approach.

Following the steps in DO-254 figure 11-1, the tool identification (1) and process identification (2) steps are straightforward. This discussion is aimed at avoiding the need for independent assessment of the tool output, so it will be assumed that the answer to question 3 in figure 11-1 is "No, independent assessment is not available." Synthesis tools, which lead to configuration of the device, are clearly design tools, so the answer to question 4 will be "Yes, this is a level A, B, or C design tool," as long as the device will be part of a level-C or higher system.

Question 5 asks about relevant service history. The reason tool qualification is being considered is the experience of flaw-free operation. However, the vast majority of this service history will be in non-airborne contexts. There are also few if any service record databases available to support tool qualification because most projects developing devices based on FPGAs are proprietary, and there is little incentive to publicize one's experiences. Therefore, step 6, establishing baseline and problem reporting, might be difficult.

Step 7, confirming that the tool produces correct outputs for its intended application using analysis or testing, would also present difficulties. These difficulties would be mitigated if a means of verifying the output of the tool on test inputs via independent assessment were available.

The conclusion is that creating the capability to compare the outputs of two independent netlist synthesis tools to provide a path toward independent assessment will be useful either as a part of the synthesis tool qualification process or because DO-254 tool qualification will be difficult, making independent assessment an essential part of verification.

5.3.2 Independent Assessment of Netlist Synthesis Tool Outputs

An interesting idea for independent assessment of the output of a netlist synthesis tools is that of a canonical form for the EDIF file. The process of compiling VHDL into a netlist, unlike the process of P&R, is not dominated by an optimization algorithm. In principle, two netlists produced by two different tools from the same VHDL source ought to be equivalent in some sense.

This is complicated by the fact that the names of variables, signals, and gates would differ, and the order in which elements appear in the file would differ. However, a canonical way to order the elements could be defined, and a canonical scheme of naming variables could be determined, such that any EDIF file could be transformed to canonical form. Then if two independent compilations yielded two EDIF files which, when transformed to canonical form were identical, there could be confidence in the results based on independent production of identical results.

Unfortunately, netlist synthesizers appear to exploit a degree of freedom to assist the later place and route steps by inserting registers and rearranging the operations (as long as the logic remains the same). So this scheme of netlist file comparison may be less straightforward than it might first appear.

5.4 QUALIFICATION OF NETLIST FUNCTIONAL SIMULATION TOOLS

Development process artifacts for netlist-level simulation tools do not appear to be available in sufficient detail to act as tool qualification data satisfying DO-330. Therefore, qualification based on the tools' development process may be impractical. The approach to tool qualification would need to be based on experience, as described in DO-254 section 11.

ActiveHDL, provided by Aldec, has a mature functional simulation tool with extensive service history. Because the format of the EDIF file specifying the netlist has not changed in many years, assessment of the functional simulation tool need not suffer from the phenomenon of multiple version changes raising questions about the applicability of the bulk of the available service history.

The three remaining concerns are:

- Obtaining service-history data in a manner that is sufficiently formal to satisfy considerations in section 11.3 of DO-254.
- Isolating or identifying the portion of service history that is involved with safety-critical applications (whether airborne, nuclear industry, biomedical, or other applications requiring a high degree of confidence in correct operation).
- Making the argument that netlist functional simulation is no different in one domain than in another. The fundamental distinction in service history is between applications with very low tolerance for risk of defects, and other applications. Therefore, successful tool experience in all low-risk-tolerance designs is relevant to airborne applications, and the body of relevant service history for DO-254 tool qualification purposes is that portion of the overall product experience data.

5.5 QUALIFICATION OF P&R TOOLS

The prospects for qualification of a given P&R tool, either by process-based assurance or based on service history, are impacted by the facts that these tools employ very sophisticated algorithms and that the "target" FPGAs that need to be routed rapidly advance to new chip generations and versions. Therefore, service histories of successful use tend to be composed of many experiences with FPGAs having different design rules than those of the current target FPGA, and relatively few involving the current target FPGA. Whereas this history can sensibly be treated as evidence of correct performance of some central, timing-rule-independent kernel of the P&R tool, the designer does not see a clean separation between that kernel and the timing-rule layer.

Moreover, the argument that the P&R is a similar activity in any other low-risk-tolerance design as it is in an airborne design is not as compelling as was the case for netlist functional simulation. The timing challenges are different in various application domains, so the issue of relevance is a serious concern. Cross-project qualification (in the DO-254 sense) of P&R tools would require sufficient service history using the specific tool version necessary to deal with the FPGA device being configured for a given design. To be considered relevant, this service history would also need to involve devices developed for application in the airborne context. Because obtaining sufficient service history meeting both criteria will be difficult, the prospects of cross-project qualification (in the DO-254 sense) of P&R tools are remote.

5.6 QUALIFICATION OF TIMING-AWARE SIMULATION TOOLS

The prospects of qualification of timing-level simulation tools are also impacted by the fact that the "target" FPGAs that need to be routed rapidly advance to new chip generations and versions. For these tools, a cleaner separation between a simulation kernel and the design-rule layer exists. That may provide a path to qualification for the kernels of some timing simulation tools.

To the extent that timing-aware detailed simulation is used only as a validation tool to expose flaws to be corrected before safety assurance requirements-based verification steps are performed, these tools do not require qualification. If timing simulation results are offered for certification credit, however, they will need to be backed up by hardware scope results confirming key features of the simulations or by other assurance techniques.

5.7 QUALIFICATION OF TOOLS ASSISTING HARDWARE VERIFICATION

When the (trial) configuration has been defined for a design and loaded into an actual FPGA, verification of correct performance of that hardware/configuration combination can proceed. If functional simulation, timing simulation, and physical testing give identical results, then this offers independent assessment that the various forms of validation and verification have worked properly. Hardware-based verification is specified, for example, in Order 8110.105 section 6-2 paragraph D. Outputs of verification tools will be offered as evidence that no fault introduced by netlist synthesis or P&R has evaded detection, so assurance that these tools function correctly is important.

The hardware-based verification step relies on various tools: test generators, ways to place signals onto the pins of the FPGA, and libraries of monitors to see the chip behavior at speed. Aldec's CTS provides a unified solution to the issues of generating the entire test bench suite of tests used

for functional simulation, placing the signals—with correct timing—on the FPGA inputs, and capturing results. Mentor Graphics provides tools for similar purposes. A compliance test system, in use for projects developed by Thales Avionics, has proven useful.

Applicants for certification need to verify correct performance in the target board or propose and justify alternate means of verification. For situations in which it is impractical to feed the entire suite of requirements-based tests into an FPGA situated on the actual board used in the avionics application, tools of this nature provide an excellent means of verification.

Vendors of these tools mention qualification for use of these tools in DAL-C and higher designs. A central concern, in the context of FAA (or EASA) certification, is whether the vendors' tool qualification data packages are acceptable as relevant and adequate evidence of tool qualification. This report recommends that a unified mechanism be defined to assist in the assessment of that evidence.

5.8 CURRENT PRACTICES REGARDING FPGA TOOL QUALIFICATION FOR FAA CERTIFICATION

Most FPGA developers approach the tool-assessment and qualification issue in the context of DO-254 by taking the assertion that the correctness of the tool output is independently verified using alternative means. The justification and acceptance of this approach appear in paragraph 3 of section 11.4.1 of DO-254. The note in paragraph 9 of that same section states that using a design tool without independent assessment of the tool's output or establishing relevant history is discouraged because it may prove to be too challenging a task.

This state of affairs has come about partly because tool assessment and qualification data, as specified in section 11.4.2 of DO-254, are often unavailable for the design tools necessary for development of the FPGA. However, some companies have begun to address this issue. One product relevant for tool qualification is ALDEC's DO254 Compliance Tool Set (DO-254/CTS™).

This compliance tool set provides support specifically geared toward section 11.4 of the DO-254 standard. The approach taken is a provision of a motherboard based on the FPGA with a PCI interface to a host computer, serving as a controller for the in-hardware simulation process. This is combined with software, including a Compliance Verification Toolset (CVT), which is a software package for in-hardware simulation, and a verification tool based on test vectors.

The CVT toolset supports the verification process presented in section 6.2 of DO-254. ALDEC has also introduced the DO-254/CTS (TM Tool Qualification Data Package). This includes a tool qualification plan, operational requirements, a tool test plan, and a qualification accomplishment summary. There are also qualification data packages for other tools (the HDL simulator and an HDL coverage tool).

The expense of these tools may be a consideration for any given development, and there as yet does not appear to be a top-to-bottom set of artifacts that facilitates qualifying every aspect of the tool suite, but the trend is clearly in the direction of better opportunities for tool qualification.

5.9 INDEPENDENT ASSESSMENT OF P&R TOOL OUTPUT

It is plausible to consider the detailed simulation step as an independent assessment of the output of the P&R tool. If approval of the design depends on that assertion, then the simulation tool should at a minimum be distinct from the tool that synthesized the P&R from the netlist to help ensure that a single source of problems does not corrupt both the design and the checking intended to expose such problems. In particular, performing the simulation as a "next step" using an integrated placement and simulation tool would significantly weaken the assertion of independence.

A much stronger form of independent assessment can in principle be achieved by obtaining two uncorrelated detailed simulation tools (either from different vendors or as two distinct products from the same vendor) and executing simulation of the verification suite of tests using both tools. Unfortunately, this involves doubling the work in one of the more time-consuming development steps and the extra expense and learning curves associated with the second simulation tool. Therefore, there is a risk that one of the "independent" tools is not applied in a comprehensive manner.

5.9.1 Qualification of Detailed Timing Simulators

The other approach to demonstrate that the detailed simulation tool has performed as intended is tool qualification. In principle, some combination of artifacts about the process of developing and testing the tool and successful service history in relevant applications can be used to demonstrate that the likelihood of an error in the simulation is sufficiently remote. Given that evidence (assessed with a stringency appropriate to the device's assigned DAL), the DERs can decide whether to accept, as part of the approval process, the simulation results.

Currently, the route of tool qualification is rarely used. This may be because of a lack of appropriate artifacts and records of service history, insufficient guidance and recommendations concerning qualification of these design tools, or a lack of experience in PLD tool qualification on the part of vendors, developers, and DERs. The ideal is to be able to demonstrate tool qualification when that is available and appropriate, or independent assessment when that can be done.

5.9.2 Treating FPGA Development From a Purely Hardware Perspective

To circumvent the burdensome and ill-defined issues that come from treating FPGA development as a software process, some chips are certified as if they were simpler hardware components. However, the concepts that a DER would apply in that situation (if the designation as simple hardware were accepted) are a poor match with the reality of the FPGA.

For example, there are DERs mandating individual equations for every pin of the devices. That is appropriate in many cases, but when a device puts out a 32-bit bus, this approach is both inappropriately detailed and repetitive, and misses fundamental issues that could lead to problems. For example, it is much more likely that a subtle error in bus protocol will occur than that the designer will cross and switch lines 25 and 26 of a data bus.

Many errors in modern complex PLD development are due to erroneous creation of the HDL statements implementing low-level requirements. Such errors are more closely akin to software coding errors than to classic hardware design errors. Therefore, it is important that the approval

process and certification guidelines include considerations that are derived from software aspects of development. This means that the tendency to treat complex FPGAs under the approval criteria applicable to hardware devices is a non-optimal means of assurance.

Still, unlike the case for pure software modules, PLD development carries risks of subtle hardware-related flaws. For example, the DER might reasonably request documentation demonstrating that the design and timing rules used and tested in the lab will be equally applicable in the aircraft environment, and this sort of issue does not arise in the context of pure software development. Because of this mixture of hardware and software aspects, neither pure DO-178C nor DO-254 assurance standards are comprehensive when applied in the context of complex PLDs. Possible hardware-related extensions or additional guidelines, applicable for approving complex PLDs, should be considered. In particular, when formulating recommendations for guidelines on qualification of PLD development tools, the issue of assuring that the tools can be made to deal appropriately with questions on the cusp of hardware and software should be addressed.

6. RECOMMENDATIONS.

The aim of these recommendations is improved safety assurance of the development of FPGA (and other PLD) configurations, particularly in the areas affected by the use of development tools. Although some recommendations point to additional assurance requirements for developers who currently may take a less uniform approach to justifying trust in tool outputs, the intent of these recommendations is to avoid imposing undue burdens and additional work without proportional gains in assurance. Some recommendations support streamlining the current development/assurance process.

The recommendations in this section are grouped by the nature of the recommendation (improved development activities, treatment of FPGA development using software assurance). The recommendations are provided in no particular order of importance.

Some of the recommendations are applicable to development of devices intended to perform functions classified at some specific DAL. As indicated in section 1.4, when these recommendations use phrases such as “at DAL-C” or “for DAL-B development,” that wording should be taken as a shortened way of saying “for development of FPGAs intended to perform design assurance level C (or B or A) functions in systems requiring certification.”

6.1 ADDITIONAL AND IMPROVED DESIGN AND VERIFICATION ACTIVITIES

6.1.1 Recommendation 1A: Treatment of Block Diagrams

If the developers express the intended chip functions in the form of tool-supported block diagram input, rather than VHDL statements, then the block diagrams should be treated as documentation of the intended behavior rather than as the source code. The VHDL generated from those block diagrams should be captured as source code and reviewed to affirm that all intended dependencies and functions are properly expressed.

6.1.2 Recommendation 1B: Review of VHDL Statements

For FPGAs filling DAL-C, DAL-B, or DAL-A functions, VHDL statements should be reviewed for avoidance of constructs that commonly lead to erroneous behavior in FPGA designs. The result of an inspection by automated tools such as ALINT® should be considered an acceptable form of review.

6.1.3 Recommendation 1C: Validation of EDIF file

For development of FPGAs filling DAL-B or DAL-A functions, the EDIF file output of the netlist synthesis tool should be validated in one of two ways: 1) evidence of tool qualification may be presented or 2) evidence of equivalence of two EDIF files produced from the same VHDL sources by two different netlist synthesis tools may be presented.

6.1.4 Recommendation 1D: Review of Reset Behavior and Clock Domains

Because a significant fraction of injected defects are related to multiple clock domains and inadequate specification of reset activities, there should be a review assessing completeness of state on resets and proper treatment of clock domains.

6.1.5 Recommendation 1E: Strengthened Validation of Gate Layout

With the exception of problems involving packaged IP blocks, no “smoking gun” instances of the place and route (“netlist compilation”) step injecting defects were encountered in the example designs studied in detail, or mentioned by FPGA engineers describing other development projects. In addition, the FPGA designers expressed the likelihood that if other developers had experienced concrete instances of errors introduced by P&R tools, they would have heard of them. However, this represents only a sampling of the engineering community, and the possibility of errors introduced by faulty P&R tool behavior cannot be rejected. In recognition of the fact that the P&R tools are more complex and perhaps less reliable than the netlist synthesis tools, validation of the layout produced by P&R tools for designs at DAL-C and higher should include additional validation or more stringent verification in one of two forms:

- Identification of timing-critical paths and validation of the layout by examining timings in simulations exercising those paths.
- Comprehensive timing-level simulation and detailed examination of a sample of the timings in the correct results produced.

6.1.6 Recommendation 1F: Evidence of Acceptability of Timing Simulations

For DAL-B designs and higher, additional evidence of acceptability of the timing simulations should be presented. Acceptable evidence is recommended as:

- Simulations completed using a qualified tool, with qualification evidence presented.
- Simulation outputs independently assessed by running multiple simulation tools and comparing samples of the results.
- Comparison of simulation results to timings observed and measured on the target hardware running the same tests as the simulations.

6.1.7 Recommendation 1G: Qualification of Test Vector Automation

If a tool is used to automate or assist in applying test vectors to the FPGA device or simulator, and this tool is the sole means of assuring test coverage, then this tool should be qualified for DAL-B and DAL-A.

6.1.8 Recommendation 1H: Cross-Verification of Timings

Because there are essential changes associated with each move to a different FPGA generation, which will make qualification of detailed timing simulators difficult, full tool qualification of detailed timing simulators is probably not feasible. At DAL-C, results of timing simulations using such tools may be accepted as verification evidence. However, in that case, at least a sample of those results should be cross-verified by examination of actual hardware timings.

6.1.9 Recommendation 1I: FEC

FEC, discussed in section 4.3.2, is an elemental analysis technique. It is the analogue in FPGA development of MCDC in the software context. For devices performing DAL-A functions, guidelines requiring FEC for FPGA safety assurance should be considered.

6.2 TREATMENT OF FPGA CONFIGURATION DEVELOPMENT AS SOFTWARE

Creation of an FPGA configuration that correctly implements the system and high-level requirements more closely resembles software development than typical hardware component development. DO-254, rather than its software counterpart DO-178C, applies to safety assurance of PLAs and FPGAs. There are aspects of the DO-178C verification criteria that are more stringent for a given DAL than those in DO-254.

It is beyond the scope of this research to make recommendations concerning changes or additions to DO-254. However, for optimal safety assurance, FPGA configuration development and verification needs to adhere to certain standards applicable to software in general.

6.2.1 Recommendation 2A: Use of DO-178C Assurance Techniques

It is recommended that guidelines for FPGA safety assurance should accept the use of certain software safety assurance objectives and activities described in DO-178C. In particular, sections 4.5, 5.2.1, 5.2.2, 5.3, 5.5, 6.3.1, 6.3.2, 6.4.1, 6.4.2, 6.4.5, and 6.5 of DO-178C contain assurance techniques or activities that are not clearly espoused by DO-254 and are suitable for application to FPGA development at DAL-C and higher.

An evaluation of the relative utility of the objectives and activities in these sections of DO-178C, for FPGA safety assurance, is provided in section 6.5 of this document.

Because standard VHDL does not include constructs analogous to the C “#ifdef” statements, VHDL statements inserted for monitoring purposes should be treated as deactivated code, per section 5.2.4 of DO-178C, rather than as dead code.

Test coverage analysis (section 6.4.4) need only cover high- and low-level requirements. Branch coverage and MCDC are not suitable tools in the FPGA context.

6.2.2 Recommendation 2B: Clarification of Use of DO-330 for FPGAs

DO-330 describes considerations applicable to software tool qualification. Although there is not perfect alignment between tools used to design FPGAs and software development tools, most of DO-330 should be considered relevant to FPGA tool qualification.

The use of DO-330 should be clarified as a standard for tool qualification of tools used in FPGA development.

6.2.3 Recommendation 2C: Timing Constraints Should Be Treated as Source Code

The timing constraints data used to guide the P&R step are an adjunct form of source. It is likely that these data will be recognized as source code if a revised DO-254 treats FPGA configuration as akin to software development. In any event, all source code management guidance should apply to the file(s) providing timing constraints and targets to the FPGA development tools.

6.2.4 Recommendation 2D: Evidence of Reproducible Configuration

In software development, the capability of reproducibly building the resulting product (executable code or library modules) is considered essential. This concept also applies to FPGA configuration development for the same reasons.

For devices performing functions classified at DAL-C and higher, evidence should be provided of reproducibility of the exact FPGA configuration, starting from change-controlled source and timing-constraint files (and if necessary, captured information about the operating system, environment, and tool versions). Configuration checksum equivalence between two independent builds should be considered adequate evidence of reproducibility.

6.2.5 Recommendation 2E: Qualification of Synthesis and Verification Tools

The following is a sensible granting of flexibility in cases where tool assurance in FPGA development is based on DO-330:

For devices performing functions classified at DAL-C and higher, for each synthesis/validation or synthesis/verification pair of steps, at least one of the synthesis tool or the verification tool should be qualified. At DAL-B and higher, the synthesis tool, which is capable of injecting defects into the design, should be qualified, as would be required per DO-330.

6.2.6 Recommendation 2F: Treatment of Unexploited Device Capabilities

Guidelines should be issued concerning how unexploited device capabilities on an FPGA (e.g., blocks for interfacing to a specific bus type when no busses of that type are used) are to be treated for safety assurance purposes. The recommended consideration for such capabilities is that they need not be certified or verified if they are not used. However, guidelines should require acceptable evidence that the presence of the unexploited capability will not affect the performance of the

verified functionality. The nature of what constitutes acceptable evidence may depend on the function performed by the device. Stringent standards should be applied for devices performing DAL-B and DAL-A functions.

6.3 GUIDANCE AIMED AT TOOL VENDORS

The recommendations in this section involve issuing, in some form, advice that tool vendors can follow to facilitate the process of qualification of mature or well-developed tools. The authors recognize that this form of advisory information is not normally directed at third-party vendors.

6.3.1 Recommendation 3A: Clarification of Path to Qualification of Tools

Clarification of the path to qualification of mature netlist synthesis and functional simulation tools should be made available to tool vendors.

6.3.2 Recommendation 3B: Qualification of Test Vector Presentation Tools

This recommendation deals with the issue of qualifying tools that present test vectors to FPGA hardware. Because the vendors intend for these to be used as qualified tools, the path to qualification should be as uniform and clear as possible.

Mechanisms should be specified by which test vector presentation tools can meet tool qualification standards. Specifications should clarify the nature of data to assist in experience-based qualifications, and advise about acceptable process-based qualification evidence.

6.3.3 Recommendation 3C: Establishment of "Experience-Tested" Status

As discussed in sections 2.3.1 and 5.5, qualification of P&R tools presents difficulties. The following might increase the prospects of qualification, or at least support some level of certification credit for evidence that the P&R tool has a history of proper operation.

Establishment of an "experience-tested" status, to be applied to combinations of chip versions and P&R tool releases, is recommended. This status could be considered for credit for FPGA designs using those tested combinations as partial evidence that the tool is qualified for the specific design project.

The authors recognize that the usual safety-assurance activities of the FAA do not include designation of specific tools or specific aspects of a tool as being qualified or as having any special status with regard to certification outside the context of a specific project. This may present significant complications in implementing this recommendation and recommendation 3D.

6.3.4 Recommendation 3D: Qualification of Tool Kernels

Some FPGA design tools, particularly P&R tools, must evolve with each FPGA generation and version. It is recommended that guidelines recognize separation of such tools into two parts: stable, tested and qualified kernels; and chip/version-specific variations.

An acceptable mechanism for qualification of the stable kernel should be specified. Guidelines should also present acceptable methods for independent assessment of the variations that fall outside the kernel qualification.

6.3.5 Recommendation 3E: Clarification of History and Defects Documentation

Guidelines should encourage collection and presentation of tool history and current tool defect status in a form that can be used in tool qualification based on relevant service experience. Clarification specifying what form of history evidence and defect reporting is suitable for tool qualification should be made available to vendors and developers.

6.3.6 Recommendation 3F: Active Collection and Presentation of Errata

Guidance concerning publication of errata is recommended. This guidance should strongly suggest active collection of errata and publication of the open issues at the earliest reasonable time. Absent such guidance, vendors often wait until a fix or workaround has been developed and then present the erratum to the public. It is more valuable, from a safety assurance standpoint, that information about all reported defects be made available to all developers immediately on verifying that the reported deficiency exists.

6.4 OTHER RECOMMENDATIONS

The remaining recommendations deal with activities that were highlighted in discussions with design engineers. These activities pose particularly high risks of defect injection or conceptual difficulties for practical safety assurance.

6.4.1 Recommendation 4A: Treat “Public” IP Function Blocks as Advisory

If using “public” IP (function blocks for which the VHDL source is widely available and source documentation is available), the FPGA designer should treat the IP code as advisory and untrusted. The developer should take control of the VHDL code as if it were locally developed and perform review activities and validation checks accordingly. This does not apply if the IP code has gone through a tool-qualification process because the qualification artifacts can be used as evidence that the source code is correct.

6.4.2 Recommendation 4B: Independent Assessment of Unqualified IP Blocks

This concerns the observation that in the experience of FPGA developers, much of the IP block code available, even from vendors of high-quality tools, has proven untrustworthy. If using unqualified IP blocks that are “opaque” (no source code access), then the outputs of such blocks must be independently assessed or validated.

6.4.3 Recommendation 4C: Analytic Verification of State Machines

If vendor-provided tools or IP blocks setting up state machines are used, results of analytic verification of the VHDL generated when state machine blocks are autogenerated should be presented.

6.4.4 Recommendation 4D: Proof of Separation for Multi-DAL FPGAs

When claiming partitioning of a higher DAL function from a lower DAL function implemented on the same FPGA, hardware-test-based demonstration of complete separation should be required.

The gate and signal lockdown-based technique described in section 4.5 should be considered suitable for demonstrating separation. This consists of implementing the high-DAL functionality alone, performing its verification tests capturing detailed timings, then implementing the full FPGA and comparing results and timings on verification tests. Note the caveat that proof of the effectiveness of partitioning is solid only if the set of high-DAL verification tests is complete with respect to the high-DAL requirements.

Other methods of demonstrating separation may be acceptable if the evidence is at least as convincing as that obtained by this method. To claim credit for an alternative method of demonstrating separation, the developer should show that the alternative method provides assurance equivalent to the recommended technique.

6.5 UTILITY OF SPECIFIC DO-178C ACTIVITIES FOR FPGA SAFETY ASSURANCE

This section responds to an observation that process burden should not be imposed on the airborne FPGA development community merely because an analogous burden is placed on the software development community. Each section of DO-178C mentioned in recommendation 6.2.1 in this document is evaluated as to whether it will materially improve FPGA safety assurance. Emphasis is given to activities that would have prevented, or detected at an earlier stage, defects that affected the FPGA projects under study.

The activities and objectives in DO-178C deemed relevant in that sense are (translated into their analogous statements about FPGA development activities):

- 5.2.2(b) Design decisions (particularly decisions about multiple clock domains and the use of state machines) should be analyzed to see that they do not compromise the overall device functionality.
- 5.2.2(f) Responses to failure conditions should be consistent with the safety-related requirements.
- 5.3.2(d) Design planning should address constraints that avoid common sources of non-obvious misbehavior, such as uninitialized VHDL variables, and uncheck use of block diagrams. The VHDL supplied to the netlist synthesis tool should be reviewed for conformity to those constraints.
- 6.3.1(a) Objective: Ensure that the system functions performed by the FPGA are defined and satisfy the functional and safety-related needs they are meant to address.
- 6.3.2(b) Objective: Ensure that each high-level requirement is unambiguous and accurate for its purpose. Ensure that the requirements do not conflict with one another.
- 6.3.2(c) Objective: Ensure that the requirements can be met using the capacity and features of the selected FPGA device.
- 6.3.2(d) Objective: Ensure that the satisfaction of each requirement can be verified by testing.

- 6.4.2.1(b) and 6.4.2.2(b) Multiple iterations of tests should be performed to verify that critical timings are met.
- 6.4.2.1(c) and 6.4.2.2(g) Test cases should be developed to exercise all normal state transitions for all state machines implemented and to provoke transitions that will not normally occur but should not lead to unacceptable behavior.

The remainder of this section considers each of the potentially applicable software development activities and objectives, asking whether it should be emphasized in the context of FPGA development.

6.5.1 Section 4.5 of DO-178C

The relevant objectives are that the development standards should disallow constructs that produce outputs that cannot be verified, and that robustness should be considered in the development standards.

The FPGA analog of the first would be that FPGA development standards should disallow outputs that can be in the "floating" (undefined output level) state. This need not be emphasized because in the projects studied, development standards already meet this objective (i.e., because the developers know not to leave output floating, an addition to guidance is not deemed needed).

The second objective appears to be present as a hook to make the statement (in DO-178C Note 1) that defensive programming practices may be considered to improve robustness. It is not relevant for FPGA guidance.

6.5.2 Section 5.2.1 of DO-178C

The activities in the Software Verification Process are that low-level requirements are developed from high-level requirements, and that derived low-level requirements are provided to the system processes, including safety.

The feedback of "derived low-level requirements" to the system designers is a good practice. However, because the FPGA is often treated as a "black box" at the next system level, this activity need not be required. No incidents were discovered of problems caused by basing the overall system design on the wrong internal design of an FPGA.

6.5.3 Section 5.2.2 of DO-178C

This is a set of seven software design process activities. Item (a) refers to low-level requirements and is not deemed relevant to FPGA development.

Item (b), placed into analogous FPGA terminology, would state that design decisions (particularly decisions about multiple clock domains and the use of state machines) should be analyzed to see that they do not compromise the high-level requirements (overall device functionality). This is important in the FPGA context and would have prevented later challenges in at least one observed case.

Item (c) discusses partitioning, which in the FPGA context has an entirely different set of implications. Items (d) and (e) concern control and data flow; these will be sound practice in many cases, but there is no need to include an artifact demonstrating that these activities were performed.

Item (f), stating that responses to failure conditions should be consistent with the safety-related requirements, seems like simple common sense. One project noted that problems arose from failure to follow this principle, so perhaps it also needs to be stated in an FPGA context.

Item (g) is just common sense feedback to other processes and need not be called out for FPGA development.

6.5.4 Section 5.3 of DO-178C

Items 5.3.1(a) and 5.3.2(a) refer to low-level requirements and are best left out of FPGA guidance.

Item 5.3.2(c) is a sensible statement that problems found during VHDL coding with the adequacy of the requirements or overall design should be fed back to the requirements process. This is important, but DO-254 already makes that point.

Item 5.3.2(d), in FPGA language, says that use of the netlist synthesis tool should conform to constraints defined in the planning process, which may include avoidance of uninitialized VHDL variables and manual checking of block diagram constructs. Because problems have occurred in this area, it is important to have this activity in FPGA development.

6.5.5 Section 5.5 of DO-178C

This software development process traceability section cuts to the heart of the matter of considering FPGA development as being akin to software development. Items (a), (b), and (c) are activities that provide traces from high-level requirements to VHDL statements implementing the FPGA and vice-versa. This section effectively forbids the insertion of functionality lying outside the agreed requirements.

Study of several FPGA development projects has revealed that such “unrequested features” do creep in, but no instances of problems arising from them were noted. In one case, such a feature made it possible to catch and correct a defect that otherwise might have made it into production.

The question of whether forbidding non-requirement-driven features is essential, harmful, or somewhere in between is outside the scope of this report. Because no instances of problems were observed in both projects considered for this study, this report does not consider section 5.5 of DO-178C to be needed for FPGA development.

6.5.6 Section 6.3.1 of DO-178C

Section 6.3.1 involves review of the requirements before embarking on the design of the FPGA. Although DO-254 devotes its section 5.1 to the development of requirements, no such review is suggested there.

Several FPGA development projects suffered delays because of incomplete, inconsistent, or inaccurate requirements. This makes items (a) through (d) relevant for sound FPGA design.

Items (e) through (g) are less applicable for FPGA development.

6.5.7 Section 6.3.2 of DO-178C

Section 6.3.2 discusses review of low-level requirements. It is not relevant to introduce low-level requirements review into the FPGA development process.

6.5.8 Section 6.4.1 of DO-178C

Section 6.4.1(a) states that selected tests should be performed in the integrated target environment because some errors are only detected in this environment. This testing step is so natural to FPGA developers that it need not be said explicitly in special guidance.

6.5.9 Section 6.4.2 of DO-178C

Section 6.4.2 discusses the spectrum of test cases. Because some FPGAs do make it into system use before the last defects are detected and corrected, adequate testing is important. Of the 11 items in this section, the following are important for FPGAs: normal range testing 6.4.2.1.(b) and (c), and robustness testing 6.4.2.2(b) and (g).

6.5.10 Section 6.4.5 of DO-178C

Section 6.4.5 deals with reviews and analysis of testing and verification. Although items (b) and (c) are certainly relevant to FPGA development, they are also common sense. Because no problems stemming from improper test procedures or improper analysis of test results were noted in the studied projects, there is no need to add a formal review and analysis burden.

6.5.11 Section 6.5 of DO-178C

Section 6.5 deals with verification traceability activities. Bi-directional traceability between the requirements and the verification test cases is an excellent way to help ensure that the testbench is complete enough to catch any defects. It is also true that some FPGA projects have suffered from gaps in the testing.

However, traceability is not the only sensible way to achieve and demonstrate adequate test coverage. Because bi-directional traceability imposes a very significant burden on the developers, this report does not consider the activities in section 6.5 of DO-178C to be needed for FPGA development.

7. CONCLUSIONS

This report described the activities performed in the course of developing field programmable gate arrays (FPGAs) and other programmable logic devices. This description was informed by in-depth data about the design flow of two FPGAs developed for airborne applications and by a broad

spectrum of data gleaned from FPGA design engineers in avionics and other fields that require a high degree of assurance of correct performance and safe operation.

This body of information provides the basis for understanding the classes of errors that can occur in the course of FPGA design and the degree of risk associated with each type of error (shown in table 1). The focus of this report is on potential flaws that could be introduced or overlooked because of defective design tool performance. It is observed that from the perspective of defect avoidance, FPGA design has more in common with software development than with design of other electronic components.

Safety-assurance considerations are presented pursuant to the goal of minimizing the risk associated with the use of unproven tools. Two aspects of this issue are explored: the prospects for tool qualification (and what can be done to improve those prospects) and safety assurance techniques relevant to the use of unqualified tools.

Based on the design experience data, and on analysis of current guidelines and practices, a number of recommendations are made. Each recommendation deals with either a need for modified or additional guidelines or recommended safety assurance practices, which will decrease the risk of introducing a defective device into airborne operation.

Among these are recommendations for improved design and verification activities, including a review of reset behavior and clock domain issues and qualification of test-vector automation. There are specific recommendations for applying assurance techniques from the software realm to FPGA design independently from a hypothetical update to the DO-254 standard. Among these recommendations is presentation of evidence of reproducible configuration. Other recommendations are made concerning areas of particular risk in current FPGA development, including the use of function blocks and requiring proof of separation for FPGAs covering functionality at multiple design assurance levels.

Many of the recommended practices are currently required by some design engineering representatives. FAA guidelines or other actions to promote some or all of the recommendations can introduce a degree of uniformity into the application of these practices.

There are recommendations for guidance aimed at tool vendors, to clarify expectations for FPGA design tool qualification. Among these are qualification of kernels of tools for which the non-kernel parts need to evolve as new chip generations emerge and establishment of an “experience-tested” status to be applied to combinations of chip versions and placement and routing tool releases. It is recognized that this form of advisory information is not normally directed at third-party vendors.

Implementation of a selected subset of these recommendations in manners consistent with the FAA guidance practices may improve safety assurance of the development of FPGA configurations, particularly in the areas affected by the use of development tools, without imposing undue burdens on the development process.

7.1 RECOMMENDATIONS WARRANTING NEAR-FUTURE GUIDANCE

Section 6 of this report does not distinguish, among the recommendations, which are considered more "urgent" or important than others. This section presents the several recommendations that (based on the studies done) would yield the most positive impact if guidance were issued (or DER training were to mention these issues) at an earlier time.

Recommendation 1D: Review of Reset Behavior and Clock Domains

Recommendation 2C: Timing Constraints Should Be Treated as Source Code

Recommendation 3F: Active Collection and Presentation of Errata

Recommendation 4A: Treat "Public" IP Function Blocks as Advisory

Recommendation 4B: Independent Assessment of Unqualified IP Blocks

8. REFERENCES

1. "STATEMENT OF WORK: Programmable Logic Device (PLD) Tool Qualification," Delivery Order #7 under DFACT-13-D-00008, PR# CT-15-0050-A2.

2. RCTA, Inc. (2000, April) 20-152, *Design Assurance Guidance for Airborne Electronic Hardware*. (RCTA/DO-254). Washington, D.C.: Government Publishing Office.

3. Fulton, R. and Vandermolen, R. (2014). Airborne Electronic Hardware Design Assurance: A Practitioner's Guide to RTCA/DO-254. Boca Raton, FL: CRC Press.

4. RCTA, Inc. (2011, December) *Software Tool Qualification Considerations*. (RCTA/DO-330). Washington, D.C.: Government Publishing Office.

5. RCTA, Inc. (2011, December) *Software Considerations in Airborne Systems and Equipment Certification.* (RTCA/DO-178C), Washington, D.C.: Government Publishing Office.

6. Certification Authorities Software Team (2006). "Clarification on the use of RCTA Document DO-254 and EuroCAE Document ED-80, Design Assurance for AEH," Position Paper CAST-27.

7. Certification Authorities Software Team (2006). "Frequently Asked Questions on the use of RCTA Document DO-254 and EuroCAE Document ED-80, Design Assurance for AEH for AEH," Position Paper CAST-28.

8. Doulos, (1999). How to Avoid Synthesizing Unwanted Latches. Retrieved from https://www.doulos.com/knowhow/vhdl_designers_guide/tips/avoid_synthesizing_unwanted_latches/

9. Institute of Electrical and Electronics Engineers (1998). “IEEE Standard for VHDL Waveform and Vector Exchange to Support Design and Test Verification,” document 1029.1-1998.

10. EASA Report. (2011). Development Assurance of Airborne Electronic Hardware, (EASA CM-SWCEH-001).

11. FAA Order 8110.105 (chg-1), Simple and Complex Electronic Hardware Guidance. (2008).