

**DOT/FAA/TC-12/21**

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

# **Qualification of Tools for Airborne Electronic Hardware**

June 2014

Final Report

This document is available to the U.S. public through the National Technical Information Services (NTIS), Springfield, Virginia 22161.

This document is also available from the Federal Aviation Administration William J. Hughes Technical Center at [actlibrary.tc.faa.gov](http://actlibrary.tc.faa.gov).



U.S. Department of Transportation  
**Federal Aviation Administration**

## **NOTICE**

This document is disseminated under the sponsorship of the U.S. Department of Transportation in the interest of information exchange. The U.S. Government assumes no liability for the contents or use thereof. The U.S. Government does not endorse products or manufacturers. Trade or manufacturers' names appear herein solely because they are considered essential to the objective of this report. The findings and conclusions in this report are those of the author(s) and do not necessarily represent the views of the funding agency. This document does not constitute FAA policy. Consult the FAA sponsoring organization listed on the Technical Documentation page as to its use.

This report is available at the Federal Aviation Administration William J. Hughes Technical Center's Full-Text Technical Reports page: [actlibrary.tc.faa.gov](http://actlibrary.tc.faa.gov) in Adobe Acrobat portable document format (PDF).

**Technical Report Documentation Page**

|   |  |  |   |   |           |
|---|--|--|---|---|-----------|
| 1. Report No.<br>DOT/FAA/TC-12/21   |  | 2. Government Accession No.                          |   | 3. Recipient's Catalog No.                                |           |
| 4. Title and Subtitle<br><b>QUALIFICATION OF TOOLS FOR AIRBORNE ELECTRONIC HARDWARE</b>   |  |  |   | 5. Report Date<br><b>June 2014</b>                        |           |
|   |  |  |   | 6. Performing Organization Code                           |           |
| 7. Author(s)<br>Brian Butka <sup>1</sup> , Andrew J. Kornecki <sup>1</sup> , and Janusz Zalewski <sup>2</sup>   |  |  |   | 8. Performing Organization Report No.                     |           |
| 9. Performing Organization Name and Address<br><br><sup>1</sup> Embry Riddle Aeronautical University <sup>2</sup> Florida Gulf Coast University<br>600 S. Clyde Morris Blvd.                      10501 FGCU Blvd S<br>Daytona Beach, FL 32114                      Fort Myers, FL 33965  |  |  |   | 10. Work Unit No. (TRAVIS)                                |           |
|   |  |  |   | 11. Contract or Grant No.                                 |           |
| 12. Sponsoring Agency Name and Address<br><br>U.S. Department of Transportation<br>Federal Aviation Administration<br>Aircraft Certification Service—Design, Manufacturing, and Airworthiness<br>FAA National Headquarters<br>950 L'Enfant Plaza, S.W.<br>Washington DC 20024   |  |  |   | 13. Type of Report and Period Covered<br><br>Final Report |           |
|   |  |  |   | 14. Sponsoring Agency Code<br>AIR-134                     |           |
| 15. Supplementary Notes<br><br>The Federal Aviation Administration Aviation Research Division CORs were Charles Kilgore and Emmanuel Papadopoulos.  |  |  |   |   |           |
| 16. Abstract<br><br>The objective of this research was to study the use and qualification of tools used in developing airborne electronic hardware (AEH) for aircraft. The AEH are custom, microcoded components, or devices used as part of the airborne system. The primary technologies include Programmable Logic Devices, Field Programmable Gate Arrays, Application-Specific Integrated Circuits, and similar circuits used as components of programmable electronic hardware.<br><br>The avionics standard RTCA DO-254 provides design assurance guidance on project conception, planning, design, implementation, testing, and supporting processes in the hardware design life cycle. In particular, details are discussed regarding the processes that must be followed in respective tools' assessment and qualification. This study seeks to identify and address potential safety issues in qualifying both hardware design tools and hardware verification tools.<br><br>The study was conducted in several steps, which included: literature search; industry survey; identification of primary safety, performance, and certification concerns; developing a plan for validating these concerns; conducting experiments with the tools; evaluating the experiments; and producing the final report. The results of this study were aimed at the determination of the major issues related to using tools that support AEH design and verification and recommendations for addressing these issues in the assessment and qualification process. |  |  |   |   |           |
| 17. Key Words<br><br>Programmable logic, Tools, Qualification, Airborne electronic hardware   |  |  | 18. Distribution Statement<br><br>This document is available to the U.S. public through the National Technical Information Service (NTIS), Springfield, Virginia 22161. This document is also available from the Federal Aviation Administration William J. Hughes Technical Center at <a href="http://actlibrary.tc.faa.gov">actlibrary.tc.faa.gov</a> . |   |           |
| 19. Security Classif. (of this report)<br>Unclassified  |  | 20. Security Classif. (of this page)<br>Unclassified |   | 21. No. of Pages<br>185                                   | 22. Price |

## TABLE OF CONTENTS

|  | Page |
|--|------|
| EXECUTIVE SUMMARY  | x    |
| 1. INTRODUCTION  | 1    |
| 1.1 Objectives   | 2    |
| 1.2 Problem Statement  | 2    |
| 1.3 Research Method  | 3    |
| 1.4 Audience   | 3    |
| 1.5 Results  | 4    |
| 1.6 Document Structure   | 4    |
| 2. BACKGROUND  | 5    |
| 2.1 Software and Hardware Relationship                                   | 6    |
| 2.2 Programmable Logic History   | 7    |
| 2.3 A Typical Airborne Electronic Hardware Development Flow              | 8    |
| 2.4 The AEH Design   | 10   |
| 2.5 Verification of AEH  | 13   |
| 2.6 Simple vs. Complex Electronic Hardware                               | 14   |
| 2.7 The AEH Tool Categories  | 14   |
| 2.8 The AEH Tools in DO-254 Framework                                    | 17   |
| 2.8.1 The DO-254 Design Assurance Guidance                               | 17   |
| 2.8.2 The DO-254 Tool Guidance   | 18   |
| 2.9 What Is a Tool?  | 20   |
| 2.10 When Tool Qualification Is Required                                 | 20   |
| 2.11 Tools' Disclaimers  | 22   |
| 3. ALTERNATIVES TO TOOL ASSESSMENT AND QUALIFICATION                     | 23   |
| 3.1 Independent Assessment of the Tool's Outputs                         | 23   |
| 3.1.1 The Meaning of Independent Assessment                              | 24   |
| 3.1.2 Independent Processes at All Phases of the Design                  | 25   |
| 3.1.3 What Happens When the Independent Assessment Results Do Not Agree? | 25   |
| 3.1.4 Independent Assessment Is a Process, Not an Event                  | 25   |

|       |   |    |
|-------|---|----|
| 3.2   | Service History                                   | 26 |
| 3.2.1 | Service History Case Studies                      | 26 |
| 3.2.2 | Service History Guidance for Hardware             | 27 |
| 3.2.3 | Service History for Design Tools                  | 28 |
| 3.2.4 | Service History vs. the Latest Technology         | 28 |
| 3.2.5 | Tool Service History Is Not Sufficient            | 28 |
| 3.2.6 | Testing Maturity Model                            | 29 |
| 4.    | DESIGN ASSURANCE                                  | 29 |
| 4.1   | Constrained Random Verification                   | 29 |
| 4.2   | Observability                                     | 30 |
| 4.3   | Derived Requirements                              | 31 |
| 5.    | SURVEY OF TOOL USERS                              | 32 |
| 5.1   | Aviation Community Survey                         | 32 |
| 5.1.1 | Survey Population                                 | 32 |
| 5.1.2 | Multiple Choice Answers                           | 33 |
| 5.1.3 | Narrative Answers                                 | 35 |
| 5.2   | Semiconductor Industry Viewpoint                  | 35 |
| 6.    | LITERATURE OVERVIEW                               | 37 |
| 7.    | CASE STUDIES                                      | 39 |
| 8.    | SAFETY ISSUES                                     | 41 |
| 8.1   | Hardware Design Error Characterization            | 41 |
| 8.2   | The FPGA's Environment                            | 44 |
| 8.3   | Timing Issues                                     | 44 |
| 8.3.1 | Synchronous Design                                | 45 |
| 8.3.2 | Synchronous Design—Multiple Clock Domains         | 45 |
| 8.3.3 | Asynchronous Designs                              | 46 |
| 8.4   | Wide Data Buses and Data Pattern-Dependent Errors | 46 |
| 8.5   | Combinational Feedback/Quasi-Digital Circuits     | 47 |
| 8.6   | Synthesis Issues—What the Tool Really Built       | 47 |
| 8.6.1 | Getting Less Than Expected                        | 48 |
| 8.6.2 | Getting More Than Expected                        | 48 |

|      |  |    |
|------|--|----|
| 8.7  | Hardware That Is Nonfunctional in Normal Operation | 49 |
|      | 8.7.1 Synthesizer Optimizations                    | 49 |
|      | 8.7.2 Gate-Level Verification                      | 49 |
|      | 8.7.3 Adding Test Circuitry                        | 50 |
| 8.8  | Radiation Effects and FPGA Architectures           | 50 |
| 8.9  | Radiation—DO-254 and DO-160                        | 51 |
| 8.10 | What Circuit Is Being Generated                    | 51 |
| 8.11 | Unused Inputs and Outputs                          | 52 |
| 8.12 | Other Considerations                               | 52 |
| 8.13 | Power Up/Reset Issues                              | 53 |
| 8.14 | What Can Be Done to Prevent Problems               | 53 |
| 8.15 | Design Issues Summary                              | 53 |
| 9.   | FINDINGS AND RECOMMENDATIONS                       | 55 |
| 10.  | REFERENCES   | 57 |
| 11.  | GLOSSARY OF TERMS                                  | 63 |

## APPENDICES

- A—Survey Questionnaire
- B—Survey Results
- C—Test Procedure
- D—Annotated Bibliography
- E—Hardware Case Study Experiments
- F—Evaluation Report for Hardware Design Tools

## LIST OF FIGURES

| Figure |   | Page |
|--------|---|------|
| 1      | The AEH Stakeholders  | 4    |
| 2      | Hardware and Software Boundary                                      | 8    |
| 3      | A Typical AEH Design and Verification Flow                          | 9    |
| 4      | Generic Design Flow for the PLD Tool                                | 12   |
| 5      | The DO-254 Tool Assessment and Qualification Process                | 21   |
| 6      | Survey Population—Type of Organization                              | 33   |
| 7      | Role of Respondents in DO-254 Projects                              | 34   |
| 8      | Factors Affecting System Safety                                     | 40   |
| 9      | Macroevaluation Model of the Tool Evaluation Process                | 40   |
| 10     | Functional Flaws Requiring Design Re-Spins                          | 42   |
| 11     | Communication Barriers That Can Prevent Clear Design Specifications | 43   |
| 12     | Ring Oscillator   | 47   |
| 13     | A TRM With Three Multipliers  | 48   |
| 14     | Flip-Flop Replication   | 49   |
| 15     | An SEU in an FPGA Using a CMOS Process                              | 50   |

## LIST OF TABLES

| Table |   | Page |
|-------|---|------|
| 1     | Tool Feature Comparison   | 16   |
| 2     | The SEU Mean Time to Error for a Large FPGA in Geosynchronous Orbit | 51   |

## LIST OF ABBREVIATIONS, ACRONYMS, AND LABELS

|       |   |
|-------|---|
| AC    | Advisory Circular                             |
| AEH   | Airborne electronic hardware                  |
| ASIC  | Application-specific integrated circuit       |
| BNC   | Bayonet Neill-Concelman connector             |
| CAST  | Certification Authorities Software Team       |
| CDC   | Clock domain crossing                         |
| CMOS  | Complementary metal-oxide-semiconductors      |
| COTS  | Commercial off-the-shelf                      |
| CPLD  | Complex programmable logic device             |
| CPU   | Central processing unit                       |
| CVS   | Concurrent Version System                     |
| DAL   | Design assurance level                        |
| DC    | Direct current                                |
| DER   | Designated Engineering Representative         |
| DSP   | Digital signal processing                     |
| DUT   | Device under test                             |
| EDA   | Electronic design automation                  |
| FAA   | Federal Aviation Administration               |
| FFPA  | Functional failure path analysis              |
| FPGA  | Field-programmable gate array                 |
| FSM   | Finite state machine                          |
| FVI   | Formal verification interface                 |
| GAL   | Generic array logic                           |
| GCLK  | Global clock                                  |
| GND   | Ground  |
| GUI   | Graphical user interface                      |
| HDL   | Hardware description language                 |
| IC    | Integrated circuit                            |
| ILA   | Integrated Logic Analyzer                     |
| I/O   | Input/Output                                  |
| IP    | Intellectual property                         |
| JAA   | Joint Aviation Authority                      |
| LED   | Light-emitting diode                          |
| LVTTL | Low voltage transistor-transistor logic       |
| MBD   | Model-based design/development                |
| NASA  | National Aeronautics and Space Administration |
| NI    | National Instruments <sup>™</sup>             |
| PAL   | Programmable array logic                      |
| PBD   | Platform-based design                         |
| PCB   | Printed circuit board                         |
| PCI   | Peripheral component interconnect             |
| PLA   | Programmable logic array                      |
| PLB   | Programmable logic block                      |

|        |   |
|--------|---|
| PLD    | Programmable logic device                           |
| QA     | Quality assurance                                   |
| RF     | Radio frequency                                     |
| RTL    | Register transfer language                          |
| SCADE  | Safety-Critical Application Development Environment |
| SEFI   | Single event functional interrupt                   |
| SEU    | Single-event upset                                  |
| SLD    | System-level design                                 |
| SMA    | Subminiature version A                              |
| SoC    | System-on-chip                                      |
| SPLD   | Simple programmable logic device                    |
| SRAM   | Static random access memory                         |
| SRPT   | Synchronous Receptive Process Theory                |
| SSN    | Simultaneous switching noise                        |
| TRM    | Triple-redundant module                             |
| TTL    | Transistor/transistor logic                         |
| UAV    | Unmanned aerial vehicle                             |
| UCF    | User Constraint File                                |
| V&V    | Validation and verification                         |
| VDD    | Label for IC power supply pin                       |
| VHDL   | Very high-speed integrated circuit HDL              |
| VIL    | Maximum voltage for low input                       |
| VIH    | Minimum voltage for high input                      |
| Vin_dc | Voltage input, dc                                   |
| Vterm  | Termination voltage                                 |

## EXECUTIVE SUMMARY

The objective of this research was to identify potential safety issues in the assessment and qualification of tools used in developing airborne electronic hardware (AEH) for aircraft. AEH consists of custom, microcoded components or devices that are used as part of the airborne system. The primary technologies include programmable logic devices (PLD), application-specific integrated circuits, and similar circuits used as components of programmable electronic hardware. While the study's focus was on the most popular subset of the PLD technology, known as field-programmable gate arrays, results will be applicable to custom, microcoded devices.

An avionics standard, RTCA DO-254 (referred to as DO-254), provides design assurance guidance for project conception, planning, design, implementation, testing, and supporting processes in the hardware design life cycle. In particular, details on the processes that must be followed for respective tools' assessment and qualification are discussed. This study seeks to identify and address potential safety issues in qualifying processes.

The research surveyed the literature, conducted a survey, and implemented hardware test cases to address concerns related to:

- When tool qualification should occur and what the alternatives to tool qualification are.
- Approaches used to qualify tools used in the design and verification of AEH for airborne applications.
- The use of a tool's service experience or service history as related to qualification.

This research supported policy and guidance development for aviation systems in a rapidly evolving field of technology that exhibits a proliferation of software tools. However, this report should not be considered as Federal Aviation Administration policy or guidance—it is research-focused and will be considered as input for future policy and guidance, as appropriate.

This study included a literature search; an industry survey; identification of primary safety, performance, and certification concerns; developing a plan for validating these concerns; conducting experiments with the tools; evaluating the experiments; and producing the final report. The results of this study are aimed at determining major issues related to the use of tools supporting AEH design and verification and providing recommendations for addressing these issues in the assessment and qualification process.

Even if the design and verification tools can prove that a design is functionally correct under all conditions that were considered, design errors in the hardware can still occur because of conditions beyond the scope of the design tools. The best way to avoid these errors is to have an experienced design and verification staff with knowledge of the tool limitations. This will allow the team to identify potential problems while still in the design phase and allow error mitigation

techniques to be incorporated. In addition, an experienced staff will understand the operation of the system under both normal operating conditions and error conditions.

## 1. INTRODUCTION.

Modern aviation systems, both airborne (e.g., avionics and engine control) and ground (e.g., radar and air traffic control consoles), exemplify safety- and mission-critical, dependable systems. These systems continue to become more complex, and they often operate in uncertain environments. Both hardware and software for such systems are developed using a variety of tools that must address the final product reliability, fault tolerance, and deterministic timing guarantees. Appropriate tools must be selected to meet the needs of a specific project. The quality of the tool and the assurance provided by the tool are critical components of the final target system certification.

This report, produced under a contract sponsored by the Federal Aviation Administration (FAA), describes research focusing on the use and qualification of the tools used to design and verify airborne electronic hardware (AEH) devices. Typical AEH components are programmable logic devices (PLD), application-specific integrated circuits (ASIC), and similar circuits used as components of programmable electronic hardware. The difference between a PLD and an ASIC is that PLDs are purchased as standard electronic parts and then altered (or programmed) to perform a specific function, whereas ASICs are developed to implement a specific function. The process of PLD programming is accomplished either by using an external dedicated device programmer or on the circuit board via in-system programming. Manufacturing an ASIC component requires an expensive design and fabrication process that does not allow the device to be reprogrammed. Once manufactured, an ASIC cannot be reprogrammed and, therefore, is not a PLD.

The primary, and most popular, PLD component types are field-programmable gate arrays (FPGA). There are three types of FPGAs, determined by the underlying technology:

- Fast, but volatile, FPGA based on static random access memory (SRAM) using advanced complementary metal-oxide-semiconductors (CMOS)
- Reprogrammable, but slower, FPGA using flash memory
- Non-reprogrammable, nonvolatile, anti-fuse FPGA with good resistance to single-event upset (SEU)

The scope of the research was limited to tools supporting the development of AEH, focusing on applications for developing FPGA that have been used, or have a potential to be used, in airborne applications. The principal document (DO-254 [1]), the three Certification Authorities Software Team (CAST) papers (CAST-27 [2], CAST-28 [3], and CAST-30 [4]) that are designed to clarify DO-254, and Advisory Circular (AC) 20-152 [5] do not specifically define what a tool is. However, paraphrasing the DO-178B [6] terminology, this research defines a tool as a computer program, or a hardware device used to help develop, test, analyze, produce, or modify another program, hardware component, subsystem, system or its documentation. The tools that will be focused on are software products widely used for the design and verification of hardware components. Hardware design and test tools, such as hardware emulators, are available for only very limited environments, such as that of a single processor operating in a single architecture.

Although this research focused on computer programs used as design and verification tools, the conclusions apply equally to computer programs and hardware devices used as tools.

The DO-254 [1] glossary defines tool subcategories:

“Design Tools—Tools whose output is part of hardware design and thus can introduce errors. For example, an ASIC router or a tool that creates a board or chip layout based on a schematic or other detailed requirement.”

and

“Verification Tool—Tools used to ensure performance against predetermined standards or requirements. These tools do not introduce errors, but may fail to detect them. For example, an analog or digital circuit simulator or an automated test that measures actual circuit performance.”

Software tools used for hardware design (creation/programming, synthesis) and hardware verification (checking, simulation, testing) are typically very complex computer programs operating as part of a comprehensive tool suite. Some tools are generic, supporting multiple hardware platforms. Often, the tool is associated with a specific hardware vendor and supports design and verification of that vendor’s family of electronic components. In all cases, the tools supporting hardware design and verification are off-the-shelf products with a handful of established vendors competing for their share of the electronic market. It should be noted that the aviation industry represents only a small fraction of the market, which includes military, aerospace, medical, communication, gaming, and consumer electronics.

## 1.1 OBJECTIVES.

The main objective of this study was to provide the Federal Aviation Administration (FAA) with input on what criteria should be used to determine when and if AEH tool qualification should occur. This research will attempt to identify contradictions and possible shortcomings in the language of the current DO-254 guidelines. Safety issues related to the use of the tools will also be identified and discussed. This objective will be achieved using literature and industry surveys, identification of primary certification, performance, and safety concerns, conducting experiments to address the identified concerns using a representative suite of tools, and evaluating the experimental results to create a foundation for addressing tool qualification concerns. Related objectives are to present and evaluate the state of the art in hardware design and verification tools and to establish a base for qualification guidelines for such tools.

## 1.2 PROBLEM STATEMENT.

Considering the current status of AEH guidance, the question to be asked is: “Why would one need to qualify programmable logic tools?” The ultimate goal of developing safety-critical systems is to provide evidence that, in addition to the functionality and quality of service requirements, the specific safety requirements have been met.

There are a variety of commercially available software tools suitable for use on AEH projects. Selecting the appropriate tool to use may be confusing due to varying functionality, the variety of platforms they serve, and the choice of different hardware description languages (HDL). The major commercial tools were created without considering the required DO-254 process, which makes it difficult to assure the correctness of a design produced by these tools.

It should be noted that tool qualification is only one component of the overall DO-254 design assurance process. Different qualification requirements are placed on tools used to design and verify systems at different design assurance levels (DAL). However, regardless of the DAL, DO-254 does not require tool qualification if the tool outputs are independently assessed.

DO-254 allows tools to be used on level D AEH without independent assessment of the outputs, and without a relevant service history. For level A, B, or C design tools and level A or B verification tools, DO-254 allows the use of a tool's relevant service history as an alternative to independent assessment of the tool outputs. This research has investigated the validity of this alternative to independent assessment.

### 1.3 RESEARCH METHOD.

The research performed for this report consists of three components. First, the literature was surveyed and an annotated bibliography was produced (appendix D). Second, a tool survey was conducted to determine how the design tools are used and to determine what problems are seen in practice. Finally, hardware test cases were designed and implemented on multiple hardware platforms to investigate the possibility of hardware errors occurring when design tool operation is correct.

### 1.4 AUDIENCE.

The report is primarily intended for use by certification authorities in the development of policy and guidance. The Designated Engineering Representatives (DER) and Aircraft Certification Office engineers directly involved in the certification process are also part of the target audience. The research outcome will likely also be of interest to program and procurement managers; to project leaders; to system, hardware, and software engineers; and to all others directly involved in DO-254-compliant AEH projects. This report attempts to identify contradictions and possible shortcomings in the language of the current guidelines. It also highlights related industry approaches toward the use of software tools for PLD. Figure 1 shows the stakeholders involved in the presented investigation. It must be noted that several industry representatives shared their valuable comments and opinions with the research team through e-mails, phone interviews, and personal contacts; their names cannot be listed for reasons of confidentiality.

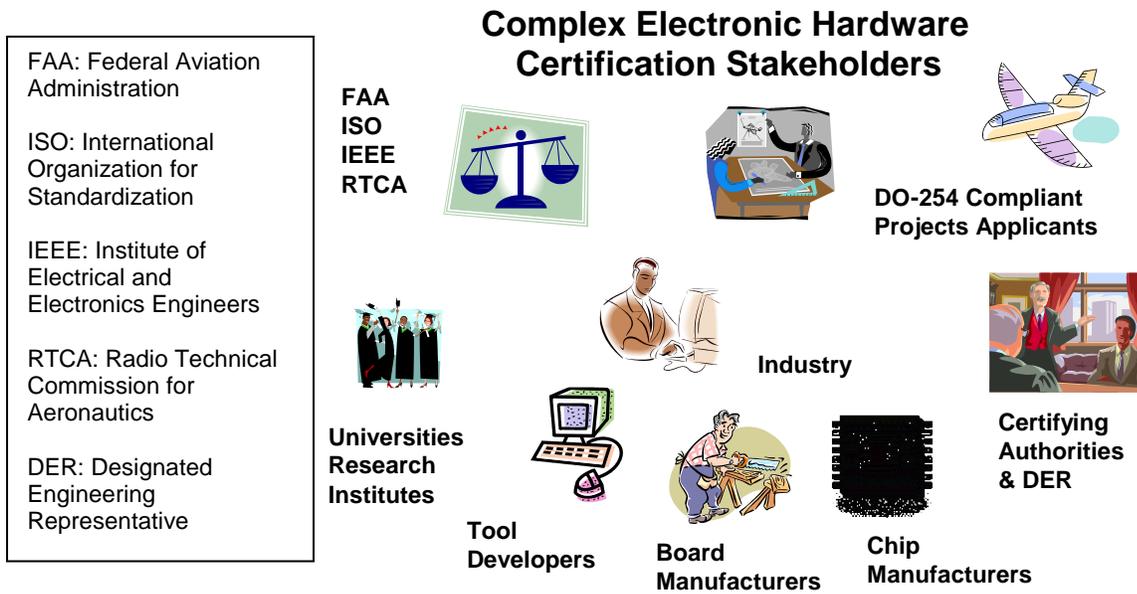


Figure 1. The AEH Stakeholders

## 1.5 RESULTS.

The authors have identified the following results, which will be discussed in detail in this report:

- DO-254 encourages the use of independent assessment and service history as an alternative to tool qualification.
- Independent assessment should not be viewed as a single event, but a process consisting of a series of overlapping independent assessments.
- Tool service history is a poor indicator of a tool's ability to produce a correct design.
- Tool qualification should be limited to the exceptionally rare case where independent assessment of the tool output is impractical or infeasible.
- Constrained random verification increases the number of errors detected by the test cases.
- Assertions can be used to increase the observability of the errors that any test case generates.

## 1.6 DOCUMENT STRUCTURE.

This report consists of 11 main sections:

- Section 1 provides introductory material, including the purpose and scope, objective, problem statement, audience, and research approach.

- Section 2 describes the AEH tool categories, and the role of DO-254 in relation to AEH tools.
- Section 3 describes alternatives to tool qualification, focusing on independent assessment of the tool's outputs.
- Section 4 examines the concept of design assurance as used by DO-254 and methods to improve design assurance.
- Section 5 presents the results of the tool use survey.
- Section 6 is a brief review of the literature compiled in the course of research.
- Section 7 presents the hardware and tool use experiments that were performed.
- Section 8 identifies a number of issues that could impact the safety of a design that has been shown to be logically correct.
- Section 9 presents the conclusions of the research.
- Section 10 provides references.
- Section 11 provides a glossary of terms as they are used within this document.

Seven appendices accompany the body of the report.

- Appendix A consists of the survey questionnaire.
- Appendix B provides details of the survey results.
- Appendix C elaborates on the experimental procedures.
- Appendix D consists of the annotated bibliography of the literature discussed in section 6.
- Appendix E provides the experimental results for the hardware test cases.
- Appendix F provides the results of the tool use experiments.

## 2. BACKGROUND.

Modern aircraft not only use increasing numbers of microcomputers and microprocessors, but also dedicated hardware, to process the growing amounts of data needed to control and monitor the status of the flight and related systems. Rapid progress of digital technology in the last 25 years can be demonstrated using an example from Airbus<sup>®</sup> industries: The number of digital units has increased from 70 to 300, the number of transistors from 105 to 108, and the number of gates per chip from 10,000 to 600,000 [7].

The recent proliferation of custom, microcoded components changed both the market and how the industry operates. These complex, programmable electronic components not only are programmed using conventional programming languages, but are developed by writing code in

an HDL used to create logic designs. The two distinctive categories used as components of programmable electronic hardware include PLD and ASIC. Often, the circuit includes dedicated processors, which is intellectual property (IP) made into the final product silicon. Most of these devices can be configured to implement a particular design by downloading a sequence of bits. In that sense, a circuit implemented on a PLD is technically software. In this report, the authors focused on software tools for hardware development in avionics and aerospace systems and methods to assure design correctness.

Software tools are used to simulate the logic, synthesize the circuit, and create the placement and routing for electronic elements and their connections in preparation for the final implementation (i.e., programming the logic devices (conventionally called “burning into the logic”). Obviously, the creation of complex digital circuits is not currently considered a software activity and is performed by hardware specialists. However, hardware and system description languages, such as very high-speed integrated circuits HDL (VHDL), Verilog<sup>®</sup>, and System C, are basically computer languages with their own syntax and semantics. The development of hardware relies significantly on the quality of tools that translate software artifacts from one form into another. Integrated programming environments allow the user to write the programs, debug the programs via simulation, convert the programs into hardware via the synthesis and place and route tools, and then debug the operational hardware. To assure the consistency of the resulting system, it is prudent that the development activity, including both software and hardware components, be done in a unified manner.

The future of software engineering for dependable, safety-critical systems is tied to the close relationship between what used to be considered separate categories: software and hardware. Because of their background and experience, software application designers focus on the development of programs to run on microprocessors and are often unaware of the possibility of implementing the system in hardware using a PLD or, the most popular technology, an FPGA. An FPGA is a prefabricated integrated circuit (IC) that can be configured to implement a particular design by downloading a sequence of bits. In that sense, a circuit implemented on an FPGA is technically software. However, circuit designers are still considered to be hardware specialists and algorithms ported to circuits are still known as hardware algorithms. Vahid [8] noted that treating algorithms implemented in circuits as “hardware” poses problems in computing system development because the hardware implementation tends to be more current than its software implementation. In addition, differences in the physical hardware implementation of an algorithm, such as using a multiport memory to support concurrency rather than using parallel dedicated memories, can dramatically affect the performance and dependability of the algorithm.

## 2.1 SOFTWARE AND HARDWARE RELATIONSHIP.

Using any combination of software and hardware in the creation of dependable, safety-critical systems requires meeting government regulations. For airborne systems installed on civilian aircraft, one needs to gain approval of the software aspects of certification through DO-178B [6], which defines the processes and artifacts to meet the approval objectives. DO-254 [1] provides a means for approval of electronic hardware components. In particular, the latter document

provides design assurance guidance on project conception, planning, design, implementation, testing, and supporting processes in the hardware design life cycle. Each of these documents addresses the issue of qualification of the tools used for creation of an airborne system in a slightly different way. It is, therefore, conceivable that a system of a specific level of safety assurance (defined by the categories from A to D, from the most to the least critical) will receive different scrutiny, depending on whether or not it is implemented in software or in hardware.

## 2.2 PROGRAMMABLE LOGIC HISTORY.

In the past, board-level digital designs consisted of large numbers of components containing a few basic gates and memory elements. Today, virtually every digital design consists of high-density IC devices. This applies to processors and memory, as well as to logic circuits, such as counters, registers, decoders, and state machine controllers. In high-volume systems, such circuits are implemented as high-density gate arrays. For prototyping or low-volume scenarios, a field-programmable approach where the software component is programmed by the end user has been more acceptable. A wide range and variety of chips makes it a daunting task for a digital system designer to research the different types of chips and to understand what they can best be used for, to choose a particular manufacturer's product, to learn the intricacies of vendor-specific software, and to then design the hardware.

In the last decade, PLDs, sometimes referred to as field-programmable devices, became an alternative to microprocessors in embedded systems. A PLD is an electronic component that, unlike a logic gate, has an undefined function at the time of fabrication and must be configured (programmed) by the end user to realize different digital designs. In the past, programmable read-only memory chips could be used to create arbitrary combinational logic functions. However, their low speed, inefficient use of space, high power consumption, and inefficiency negate the use of read-only memory in applications of a more serious nature. The introduction of simple PLD (SPLD), in the form of programmable logic arrays (PLA) in the late 70s, was followed by programmable array logic (PAL), generic array logic (GAL), and subsequent miniaturization with the introduction of complex PLD (CPLD) that could replace an entire circuit board with several SPLDs and hundreds of thousands of logic gates. A CPLD combines a logic device and a memory device consisting of one or more programmable sum-of-products logic arrays feeding a small number of clocked registers. Most CPLDs are electrically programmable, erasable, and nonvolatile. CPLDs are less flexible than FPGAs, but have the advantage of more predictable timing and a higher logic-to-interconnect ratio.

The invention of gate array technology with a grid of logic gates that could be field-programmable gave birth to the FPGA; currently, the most popular component for the creation of complex digital designs. An FPGA is a semiconductor device containing programmable logic blocks and programmable interconnects. The blocks can be programmed to act as basic logic gates or more complex combinational functions (decoders, adders, etc.). Typically, the FPGA logic blocks include memory elements, from simple flip-flops to registers, and more complete blocks of memory. FPGA architectures are dominated by interconnect, making them flexible in terms of the range of practical designs. However, they are also far more complex, which makes assuring design correctness far more difficult.

Fully custom-made ASIC can be very expensive and time consuming to produce; however, they provide the benefits of increased density, reduced area, and high performance. A popular technique is cell-based ASIC design, which incorporates well-defined and simple functional blocks. These blocks speed up the synthesis process, as well as reduce the development time. Recently, structured ASIC includes predefined standard layers (e.g., with power, clock, testing utilities), leaving only part of a silicon mask to be custom designed. A category called system-on-chip (SoC) has a large number of functions integrated within a single device. Figure 2 presents the relationship between the technologies, demonstrating the thin boundary between hardware and software. Because the boundaries are fluid, the figure illustrates just one way of many for visualizing the problem.

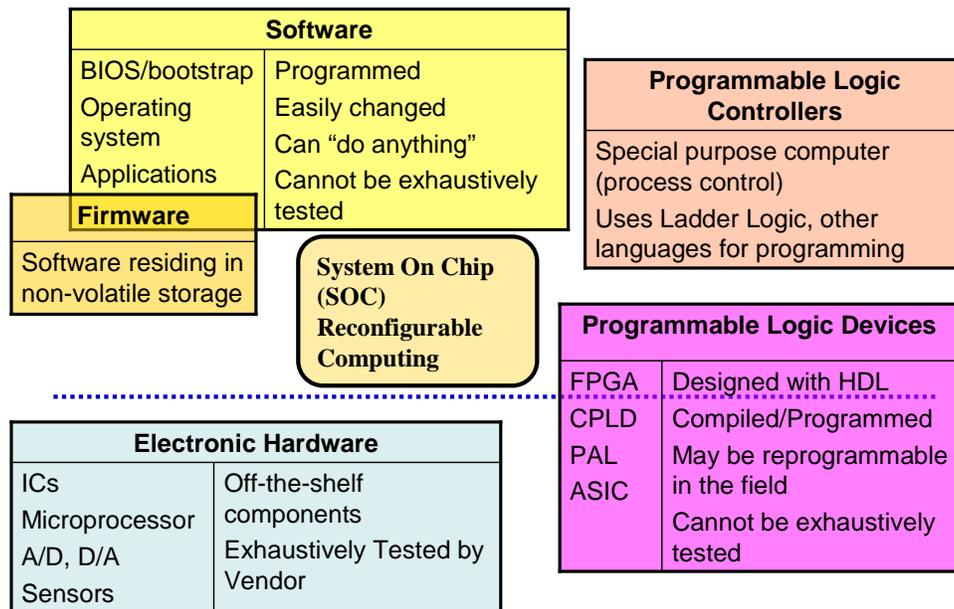


Figure 2. Hardware and Software Boundary [9]

To increase the complexity, CPLDs often include a microprocessor core with a fixed function and/or a dedicated functionality containing a specified IP core. Such an IP core, referred to as a soft core, is described by its logic function and expressed in an HDL for easy integration with other functionality. Effective modern PLD can be configured to provide multiple embedded microprocessors within a logic fabric.

### 2.3 A TYPICAL AIRBORNE ELECTRONIC HARDWARE DEVELOPMENT FLOW.

A typical hardware design flow is shown in figure 3. The design team receives the hardware requirements and then designs a system meeting those requirements. The designers use simulators to debug the design, and also to verify that the design meets all the logical and timing requirements. Any errors in the design that can be detected through simulation, such as logical and timing errors, are identified and corrected. Hardware designed using this process meets the hardware requirements as interpreted by the designers.

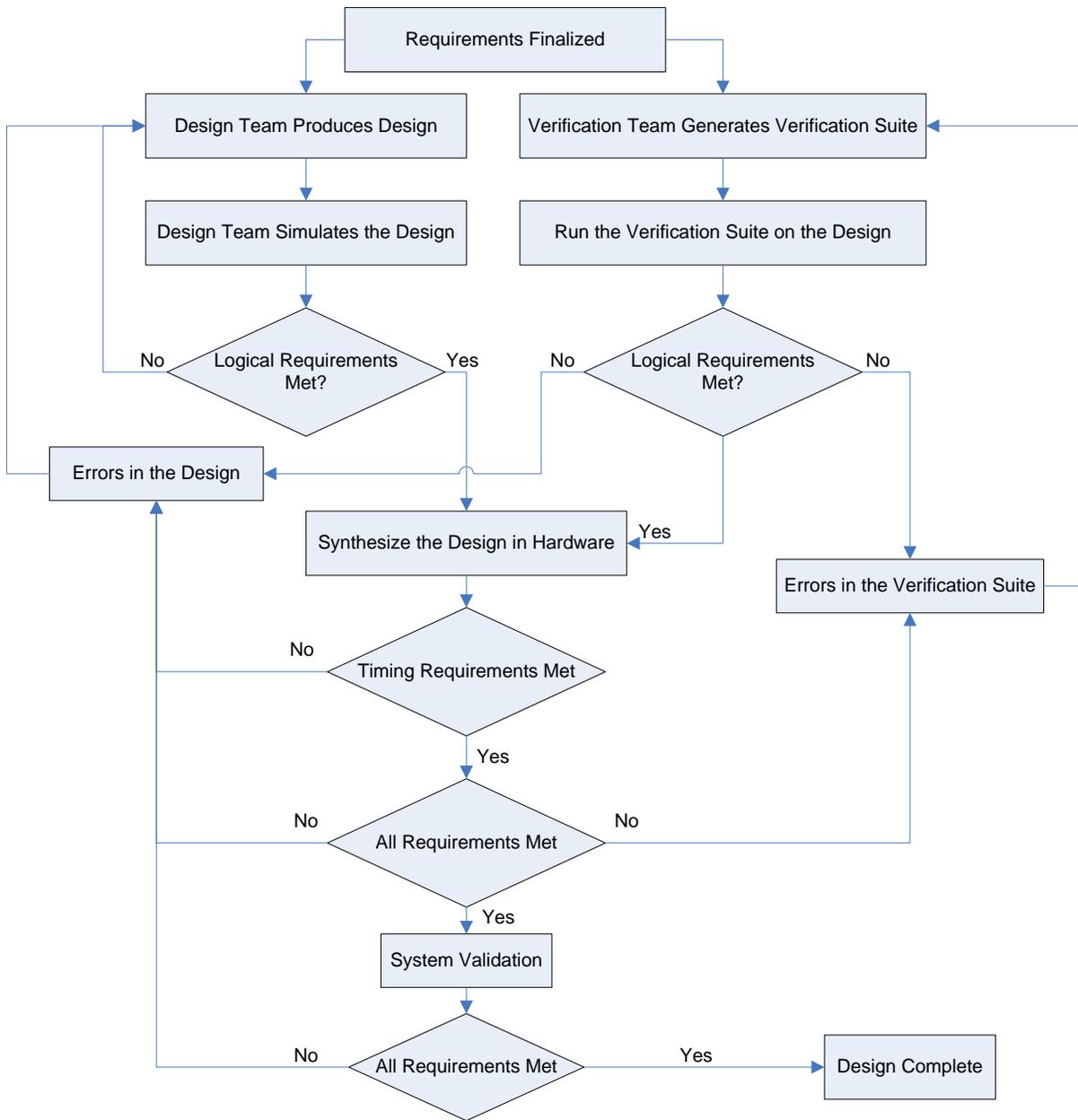


Figure 3. A Typical AEH Design and Verification Flow

The verification team begins development of the verification suite in parallel with the designers using the same set of requirements. The goal of the verification suite is to provide an independent assessment of the correctness of the design. In general, the verification suite is limited to verifying the logical correctness of the design because verification tools have limited abilities to address timing-related errors. After the verification suite is run on the design, there are three possible causes for any errors that are identified:

1. The design is in error. The design must be corrected and the verification suite must be run on the design again.
2. The verification suite is in error. The verification suite must be corrected and the verification suite must be run on the design again.
3. Neither the design nor the verification suite is in error but they expect different results. This is usually the result of incomplete or vague hardware requirements. In this case, the hardware requirements must be corrected. The design and/or the verification suite are adjusted to reflect the new requirements and the verification suite must be run on the design again.

Running the design through the verification process allows problems in the hardware requirements to be identified. Because flawed requirements can produce an incorrect design, it is desirable to detect problems with the hardware requirements as soon as possible. In many design flows, the verification suite is run against the design every night.

Once the design passes the verification suite, the design is then synthesized and implemented in hardware. The timing performance of the hardware is verified against any timing requirements and is also compared to timings predicted by the simulation. Hardware problems, such as power integrity, signal integrity, and noise problems, are extremely difficult to simulate, so it is common for hardware performance to differ from simulated performance. Variations from the predicted timing may result in the hardware producing logical errors. It is desirable to evaluate the hardware timing using as robust a test suite as possible. Some designers will use hardware-in-the-loop techniques to run the full verification suite on the actual hardware.

Unlike logical requirements, timing requirements are rarely a pass/fail decision. A timing requirement specifies minimum and maximum propagation delays, but not all timings within the specification range may be equal. Although all the allowed timings may produce correctly functioning hardware, some timings may result in robust hardware while others may produce hardware with minimal timing margins. An experienced designer is needed to determine the optimal timing.

Finally, after the design meets all timing and logical requirements, system validation is performed to assure that all system-level requirements are met when the design is used as part of a full system.

## 2.4 THE AEH DESIGN.

The previous section discussed the AEH development process. This section will focus on the design side of the development process. Logical design entry may be accomplished in three ways: (1) creating a schematic diagram with a graphical computer-aided design tool, (2) using a text-based system to describe a design in an HDL, or (3) a combination of the graphical and textual methods. The initial logic entry, however it is performed, is usually not optimized. Because the initial design entry might not be optimized, dedicated algorithms are used to

optimize the circuits. Once the circuits are optimized, additional algorithms are used to analyze the resulting logic equations for the purpose of synthesizing the circuit to fit the design into the PLD. Simulation is used to verify correct operation of the circuit, often requiring the user to modify the initial design entry to correct errors. When a design can be successfully simulated to verify the correctness of its simulated behavior, it can be loaded into a programming unit and used to configure the PLD. It is critical to note that, after the original design entry step and any required design entry corrections performed manually by the designer, all steps are performed automatically by software tools.

The more complex programmable hardware components become, the more complex and sophisticated the tools supporting development and verification of the design must be. For complex devices that can accommodate large designs, a mixture of design entry methods for different modules of a complete circuit can be used. For example, some module designs might be described using a low-level circuit description language like ABEL, others might be described graphically using a symbolic schematic capture tool, while still others might be described using a full-featured HDL such as VHDL or Verilog. These languages operate using variables and hardware signals in addition to sequential constructs, including a variety of concurrency constructs that specify parallel implementation reflecting the nature of digital circuits. The software necessary for performing these tasks is supplied by either the hardware manufacturer or a dedicated third-party tool vendor.

For FPGAs, additional tools are required to support the increased complexity of the IC. The device-fitting step includes mapping from basic logic gates into the FPGA logic blocks, placement to select specific FPGA blocks to use, and a router to allocate the wire segments to interconnect the logic blocks. With this added complexity, the tool might require a fairly long period of time (often more than several hours) to complete the design.

Software tools for embedded system development, including that of AEH, are used for two different reasons: the creation of the system hardware, and the development of software that runs on the processors included in the system. This report focuses on the creation of system hardware. The tools used for hardware design may be applied to a variety of functions, including circuit synthesis, logic circuit and hardware simulation, timing analysis, and physical synthesis. Another aspect of the hardware creation activity is verification. Logic design verification addresses most types of design debugging at every point in the design flow, from static timing analysis to support for equivalency checking and formal verification. Functional verification addresses the syntax and functionality at the design level using HDL analysis, simulation, and test bench generation. Timing verification uses the static timing and delay calculations.

Software tools are critical for the implementation of AEH circuits and devices. To design any modern device, one must use a suite of sophisticated tools including (at a minimum) simulation, synthesis, and place-and-route. Such tools are typically made available by an entity external to the developer. Simulation is supported by accessible and cost-effective tools; however, place-and-route tools are tightly connected to the specific hardware silicon architecture and vendor. In the middle of this hardware development cycle is logic synthesis. The front-end of the logic synthesis problem is very complex and not specific to any silicon architecture, while the back-

end stages of synthesis are architecture-specific. A sophisticated technology for parsing, elaborating, and inferring conceptual logic design from code written in an HDL—such as VHDL, Verilog, or SystemC—facilitates both the creation of the desired digital logic circuit design and the eventual mapping into an architecture-specific physical layout.

PLD manufacturers provide automated tools that facilitate this design flow. For creating the hardware circuitry, these tools allow the user to build a system using predesigned building blocks for processors, memory controllers, dedicated processing circuits (such as for digital signal processing), and communication modules (such as for universal asynchronous receivers/transmitters). The software allows easy instantiation of these subcircuits and can automatically interconnect them on an FPGA chip. A generic design flow is shown in figure 4.

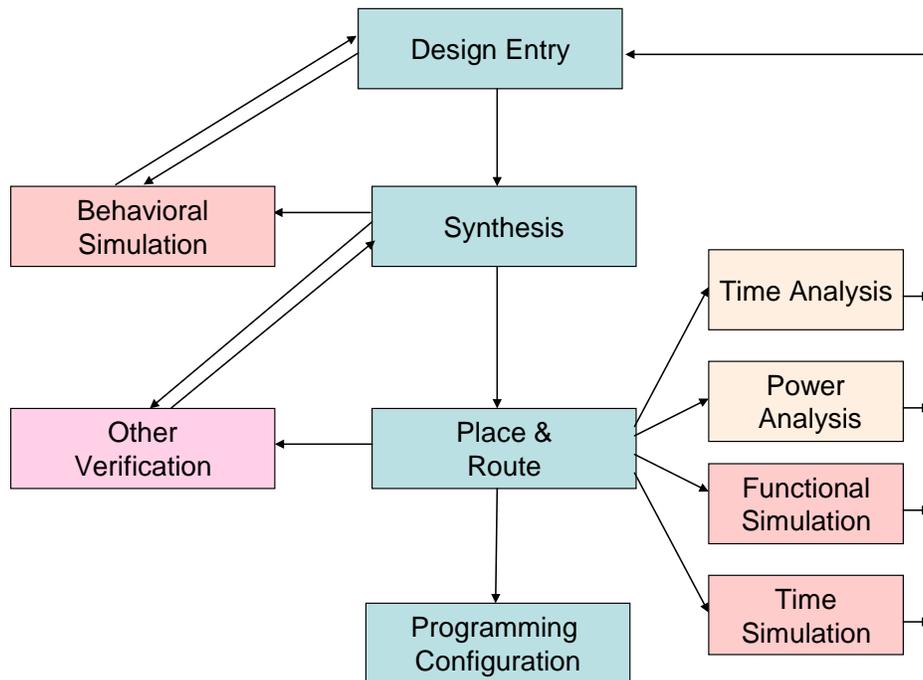


Figure 4. Generic Design Flow for the PLD Tool

Normally, it takes years of real customer designs and exercising a tool to find the peculiarities and deficiencies in the tool algorithms and then tuning the algorithms to achieve a technology that consistently delivers high-quality results over a wide range of AEH architectures and applications. Commercial synthesis tools, available from third-party vendors, achieve far better results. This directly impacts the competitive performance and utilization of the hardware supplier's silicon. Therefore, hardware suppliers have been forced into a situation where they must partner with software tool vendors to remain competitive. Some FPGA vendors offer proprietary synthesis tools as part of their low-cost (or free) tool suites, while continuing to partner with commercial tool companies for higher-end solutions, or offer customized versions of commercial tools as part of their tool suite.

## 2.5 VERIFICATION OF AEH.

Verification of a hardware design can find errors at a variety of levels. Errors can be identified in the hardware requirements, translation of the design into register transfer language (RTL), or implementation of the RTL in hardware. The first two cases involve errors made by the designer and are outside of the scope of this work, which focuses on tool qualification. Verification of the hardware implementation can be achieved using formal verification methods to prove correctness and by using simulation to verify that the hardware operates as intended. It is important to understand that formal verification only proves that the design is correct, assuming the conditions and constraints used for the formal verification. Formal verification is not equivalent to exhaustively testing a device.

Simulation requires the generation of appropriate test vectors and is an accepted traditional method for functional verification during the design creation phase. Verification of the hardware using simulation may consist of both directed test vectors and randomly generated vectors. This method is entirely adequate to verify that the design specified in RTL performs the intended function within the limits of simulation. However, verification of million-gate designs would require that transitions on every gate be tracked, resulting in a runtime of weeks for substantial million-gate designs.

Since an RTL design can be implemented in a variety of ways at the gate level, the number of test vectors grows exponentially during verification. Any unintended effect of synthesis or timing optimization can insert a design error affecting a part of the circuit, and thus manifest itself with a few combinations of values on the inputs. To guarantee detection of such an error with gate-level simulation, every possible combination of inputs must be applied, resulting in an infeasible size of test vector being required to ensure 100% error coverage. One suggested solution to this problem has been the use of formal methods [8]. The approach used is based on rigorous verification of RTL as an input artifact, while showing that the transition to the gate level is consistent, is correct, and does not change the semantic properties of the original input artifact.

One such approach to assuring the transition to the gate level is correct is to use an equivalence checker. An equivalence checker uses static verification techniques to prove that the two consecutive representations of digital design are an exact functional match (e.g., RTL-to-gate comparison after synthesis and gate-to-gate comparison after place-and-route). Gate-level simulation for modern million-gate designs is infeasible. A formal checker, an example of Other Verification (see figure 4), uses a formal verification interface (FVI) file generated by synthesis as a basis for comparison with gate-level netlists generated as a result of the first synthesis and subsequent place-and-route processes. FVI is a readable text file, including setup information with file names, paths, constraints, and name matching. If the equivalency of these representations is assured, strong evidence exists that the final design is consistent with the original design intent.

Equivalence checking provides a method of assuring the correctness of the transitions from RTL description to the physical implementation, by confirming that transformations throughout the

design flow comply with the original functionality. Equivalence checking cannot replace timing analysis. Static timing analysis tools still must be used to confirm gate-level timing.

Despite the obvious advantages of the formal equivalence checking approach, there are limitations. Some types of multipliers do not solve completely and memory blocks must be black-boxed to avoid lengthy processing times.

## 2.6 SIMPLE VS. COMPLEX ELECTRONIC HARDWARE.

DO-254 mainly concerns itself with the design assurance of AEH. DO-254 distinguishes between simple and complex hardware. A simple hardware item is defined as an item with “a comprehensive combination of deterministic tests and analyses appropriate to the design assurance level” that ensures “correct functional performance under all foreseeable operating conditions with no anomalous behavior.” [1]

A complex item of hardware is one that is not simple. Since the complexity may be a function of interconnectedness, a collection of simple items may itself be complex. For complex items, the proposed method for design assurance should be agreed to by the certification authority early in the lifecycle. AC 20-152 [5] identifies DO-254 [1] as one acceptable means for gaining design assurance approval for complex, custom, microcoded components.

## 2.7 THE AEH TOOL CATEGORIES.

The leading PLD hardware vendors are Altera<sup>®</sup>, Xilinx<sup>®</sup>, Actel<sup>®</sup>, and Lattice<sup>®</sup>. Some vendors offer internally developed synthesis tools, while others partner with synthesis tool vendors such as Synopsys, Altium<sup>®</sup>, Cadence<sup>®</sup>, Mentor Graphics<sup>®</sup>, or Synplicity<sup>®</sup>.

Contemporary logic design includes a variety of technologies:

- Design Entry: Performs HDL, schematic entry, and integration of IP cores.
- Linting: Enforces coding style.
- Synthesis: Translation of the HDL design definition into the logical primitives available on the hardware platform.
- Place-and-route: Locating and routing the hardware primitives to meet timing constraints.
- Verification: Design verification ranging from simulation to static timing analysis to equivalency checking via formal verification.

An initial review of the AEH tools market identifies a wide variety of products used in different configurations with functionality covering the entire hardware design and verification spectrum. To better understand the state of the industry, several tools were identified during the course of

the research. Table 1 shows a comparison of key features for 21 popular tools, categorized by function. The design functions match the five basic functionalities (design, entry, linting, synthesis, and place-and-route), but the verification function has been separated into several categories. Simulation tools used for both digital and analog/mixed signal simulations fall under both the design and verification categories. Verification has been divided into three categories: assertion based, test bench automation, and coverage-based tools. Assertion-based tests use assertions—checks that are coded into the HDL and are checked continuously during all simulations. Test bench automation assists in automatically generating test benches that meet a given coverage criteria. Coverage-based tests determine the number of lines of HDL code that have been exercised in a test; each line of HDL should be exercised at least once to achieve full statement coverage. To meet DAL A safety requirements, coverage methods must be extended to determine the coverage on the gate-level netlist produced by the synthesis, estimating the number of gates that were exercised by the test.

Almost all tools supporting FPGA design also support other types of programmable devices and occasionally ASIC. However, there are selected high-end tools that are ASIC-specific. Manufacturing an ASIC, because of the need for the creation of a permanent mask, is an extremely expensive proposition and is undertaken only if (1) there is a high-demand market for the device, so it can be sold by the millions (e.g., dedicated circuitry to mobile phones) and (2) the design is verifiable. To reduce or eliminate potential defects, the tools used for creation and verification of ASIC are very expensive and top of the line. The low-demand market, in contrast, calls for the use of technology that can easily be reprogrammed; therefore, tools supporting that market are less expensive and developers can afford an occasional miss in a non-safety-critical environment. However, these tools need to be carefully examined if they are to be used for a safety-critical application.

As shown in table 1, the top six tools contain all the technologies listed above, and any one of these tools can provide a full design and verification suite. Experimentation with the tools indicated that the differences between the tools vary significantly at the user interface level, but all the tools have similar design entry and synthesis capabilities. More significant than the variations in the tool suites is variation in the underlying hardware. This research chose to focus on the two largest hardware vendors: Xilinx and Altera.

Xilinx's ISE<sup>®</sup> and Altera's<sup>®</sup> MAX+PLUS<sup>®</sup> tool suites were installed in the contractor laboratory, and the personnel familiarized themselves with their operation. These tools were chosen to support the experimental portion of this work. More advanced tools from Mentor Graphics (HDL Designer<sup>®</sup>, Questa<sup>®</sup>, and O-In<sup>®</sup> Formal Verification) were too specialized to fit into the flow of this research, but they may warrant further study in the future.

Table 1. Tool Feature Comparison

|                           | Design          |         |                 |           | Design and Verification |                         | Verification    |                       |                |
|---------------------------|-----------------|---------|-----------------|-----------|-------------------------|-------------------------|-----------------|-----------------------|----------------|
|                           | Design Creation | Linting | Place and Route | Synthesis | Digital Simulation      | Mixed Signal Simulation | Assertion Based | Test Bench Automation | Coverage Based |
| MAX+PLUS II®              | •               |         | •               | •         | •                       |                         |                 |                       |                |
| Quartus II®               | •               |         | •               | •         | •                       |                         |                 |                       |                |
| ISP LEVER®                | •               |         | •               | •         | •                       |                         |                 |                       |                |
| ISE® Foundation           | •               |         | •               | •         | •                       |                         |                 |                       |                |
| Platform Studio EDK       | •               |         | •               | •         | •                       |                         |                 |                       |                |
| Actel® Designer           | •               |         | •               | •         | •                       |                         |                 |                       |                |
| HDL Designer™             | •               | •       | •               | •         |                         |                         |                 |                       |                |
| Allegro® Design Entry HDL | •               | •       | •               | •         |                         |                         |                 |                       |                |
| Alint™                    |                 | •       |                 |           |                         |                         |                 |                       |                |
| Leda®                     |                 | •       |                 |           |                         |                         |                 |                       |                |
| NC-Verilog®/NC-VHDL       | •               |         |                 |           | •                       |                         |                 |                       |                |
| Active HDL™               | •               |         |                 |           | •                       |                         |                 |                       |                |
| ModelSim®                 | •               |         |                 |           | •                       |                         |                 |                       |                |
| VCS™/Scirocco™            | •               |         |                 |           | •                       |                         |                 |                       |                |
| TauSim                    | •               |         |                 |           | •                       |                         |                 |                       |                |
| Advance MS                |                 |         |                 |           | •                       | •                       |                 |                       |                |
| Synplify®                 |                 |         | •               | •         |                         |                         |                 |                       |                |
| Questa®                   |                 |         |                 |           |                         |                         | •               | •                     | •              |
| Riviera                   |                 | •       |                 |           |                         |                         | •               |                       | •              |
| Incisive®                 | •               |         |                 |           |                         |                         | •               | •                     | •              |
| 0-In Formal Verification  |                 |         |                 |           |                         |                         | •               |                       | •              |

## 2.8 THE AEH TOOLS IN DO-254 FRAMEWORK.

Until recently, it was conceivable to verify avionics hardware using only systems-level testing, because of its simplicity. Since DO-178B required extensive effort for software assurance, whereas no assurance process was required for hardware, a large portion of system functionality migrated from software to hardware implementations to avoid certification effort. In recent years, the introduction of high-performance, programmable logic has allowed the functions of thousands of individual hardware devices to be integrated into a single hardware element. This increase in both density and complexity has allowed what used to be entire systems to be implemented in a single hardware device. Hardware devices can now be applied to complex problems that previously required software solutions. It is commonly accepted that hardware and software are closely linked and that high assurance of both is required for system reliability. The DO-254 design assurance standard is the hardware community's equivalent to DO-178B for software development assurance.

### 2.8.1 The DO-254 Design Assurance Guidance.

Two documents provide guidance for the development of dependable systems; SAE ARP 4754 [10] is the source of development guidance for highly integrated aircraft systems, while SAE ARP 4761 [11] identifies safety assessment methods to be used in the hardware design assurance process. The application of these documents allows system engineers to determine system criticality, and thus identify the DAL as allocated to hardware. Because DALs are based on classification of the worst system failure conditions, they are similar to ARP 4754 [10] "Development Assurance Levels" and DO-178B [6] "Software Levels."

DO-254 [1] was released in 2000, addressing design assurance for AEH. The guidance is applicable to a wide range of hardware devices, including integrated technology hybrid and multichip components; custom, programmable, microcoded components; circuit board assemblies (also known as printed circuit boards (PCB)); and entire line replaceable units. This guidance also addresses the issue of commercial off-the-shelf (COTS) components. The document's appendices provide guidance for data to be submitted, including independence and control data category based on the assigned assurance level, description of the functional failure path analysis (FFPA) method applicable to hardware with DALs A and B, and discussion of additional assurance techniques, such as formal methods, to support and verify analysis results. The FFPA can be accomplished on four levels—system, hardware, circuit, and component—and is used to determine which paths to analyze with increased rigor.

Because it is easily automated, elemental analysis is one of the most popular techniques to assess coverage. Elemental analysis depends on the hardware element type and complexity, and the functional operations of the element. This analysis may show either that all the low-level primitive blocks, such as counters, registers, multiplexers, adders, op amps, and filters, have been adequately tested, or that all groups of interconnected primitives have been adequately tested and achieve the verification coverage criteria. The analysis criteria of the test procedures should be based on an assessment of element operation and its integration with other elements to perform the next higher hierarchical-level hardware function. Applications of formal methods are most effective during structured portions of the design, such as during requirements capture and high-

level design, where they are effective at identifying incomplete specifications. Formal methods may be applied to verify system functionality or they may be used to confirm that a design does not exhibit certain undesirable properties, rather than to prove that it has full functionality. Although the same number of objectives is applicable to items of DALs A and B, level A may require additional design assurance techniques to provide complete mitigation of potential failures and anomalous behaviors.

One acceptable method for AEH approval is compliance with AC 20-152 [5], which refers the hardware developers to guidance of DO-254 [1]. AC 20-152 [5], which was released in 2005, applies to manufacturers and installers of products or appliances incorporating complex, custom, microcoded components, as discussed in this report, with hardware DALs from A to D. AC 20-152 provides a method for obtaining FAA approval by demonstrating that the equipment design is appropriate for its intended function. Additionally, AC 20-152 helps satisfy airworthiness requirements when these types of electronic components are implemented. It is applicable for technical standard order, type certificate, as well as parts manufacturer approval. The AC is limited to complex, custom, microcoded devices of DALs A, B, C, and D. However, an applicant does not need to show artifacts to the FAA for level D.

Because of the increasing complexity of modern digital systems, automated tools are widely used. Assessment of a tool resulting in basic tool qualification allows developers to substantiate claims regarding the tool's correctness. For qualification of a tool on DALs A and B, the qualification process is more rigorous.

### 2.8.2 The DO-254 Tool Guidance.

It is widely recognized that in safety-critical applications, with millions of gates on a chip, the role of hardware design tools and hardware verification tools becomes increasingly critical. The process of developing AEH is described in DO-254, as is the tool qualification process. Section 11.4 of this standard [1] distinguishes between design tools and verification tools:

“When design tools are used to generate the hardware item or the hardware design, an error in the tool could introduce an error in the hardware item.”

“When verification tools are used to verify the hardware item, an error in the tool may cause the tool to fail to detect an error in the hardware item or hardware design.”

Therefore, it is essential that tools be evaluated before their use, as they are critical to overall system safety. DO-254 [1] specifically states: “Prior to the use of a tool, a tool assessment should be performed.” Furthermore, it states:

“The purpose of tool assessment and qualification is to ensure that the tool is capable of performing the particular design or verification activity to an acceptable level of confidence for which the tool will be used.”

Finally, DO-254 identifies a process for “Design and Verification Tool Assessment and Qualification” [1], which is itemized in ten steps (see figure 11-1 of reference 1):

1. Identify the Tool. This includes the name, source, version, and the host environment.
2. Identify the Process the Tool Supports. This concerns the distinction between the design and verification processes, as well as the outputs the tool produces in the hardware design life cycle.
3. Is the Tool Output Independently Assessed? If the tool output is independently assessed, then no further assessment is necessary and the process is completed (Step 10 below); otherwise the assessment proceeds to Step 4.
4. Is the Tool Output a Level A, B or C Design Tool, or a Level A or B Verification Tool? If not, no further assessment is necessary and the process is completed (Step 10 below); otherwise the assessment proceeds to Step 5.
5. Does the Tool Have Relevant History? If so, no further assessment is necessary and the process is completed (Step 10 below); otherwise the assessment proceeds to Step 6.
6. Establish Baseline and Problem Reporting for Tool Qualification.
7. Basic Tool Qualification. This step seeks confirmation, using either analysis or testing, that the tool produces correct outputs for its intended application.
8. Type of Tool and Level? If the tool is a Level C design tool or a Level A or B verification tool, then the process is considered completed; otherwise (Level A or B design tool), the assessment proceeds to Step 9.
9. Design Tool Qualification. This step proceeds according to strategies described in appendix B of DO-254, DO-178B for software development tools, or other means acceptable to the certification authority. It is essential to note that “Independence of this activity from the tool development,” is called for.
10. Complete. This step relies on documenting (1) the tool assessment, (2) justification for the assessment decisions, and if applicable, (3) tool qualification data, as necessary to support the tool assignment and qualification. Section 11.4.2 of DO-254 specifies further, what tool assessment and qualification data should include.

The above process still leaves room for interpretation and is the source of numerous disagreements on whether tool qualification is required.

## 2.9 WHAT IS A TOOL?

DO-254 [1] provides guidance for design assurance of AEH defining design assurance, life cycle, processes (planning, design, validation and verification, configuration management, assurance, certification liaison), and life cycle data. However, there is no clear definition of a tool in DO-254 [1] or associated CAST papers; according to DO-178B [6], a tool is:

“A computer program or a hardware device used to help develop, test, analyze, produce or modify hardware component, subsystem, system or its documentation.”

For this purpose, a tool reduces, eliminates, or automates the objectives of the design or verification process. This very broad definition of a tool requires that the tool assessment and qualification process detailed above must be considered for design and for testing aids that may not be recognized as tools.

For example, consider a company working on a DAL A project that writes a simulator-based test bench that produces a pass/fail output. The following are observations about this test bench:

- The test bench automates the verification process and is therefore a tool.
- A verification tool for a DAL A must use the Design and Verification Tool Assessment and Qualification procedure.
- No relevant service history would exist.
- Unless the test bench outputs are independently assessed, the tool will need to go through basic qualification.

## 2.10 WHEN TOOL QUALIFICATION IS REQUIRED.

Figure 5 shows the tool assessment and qualification process flow chart. Tool qualification is required only if all three of the following conditions exist:

1. There is no independent assessment of the tool’s outputs.
2. The tool is used for levels A, B, and C for design or levels A and B for verification.
3. No relevant service history exists.

DO-254 offers the following guidance on independent assessment of the tool’s outputs:

“Independent assessment of a design tool’s output that is generated in whole or in part by the tool 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 upon the correctness of the design tool output alone.”

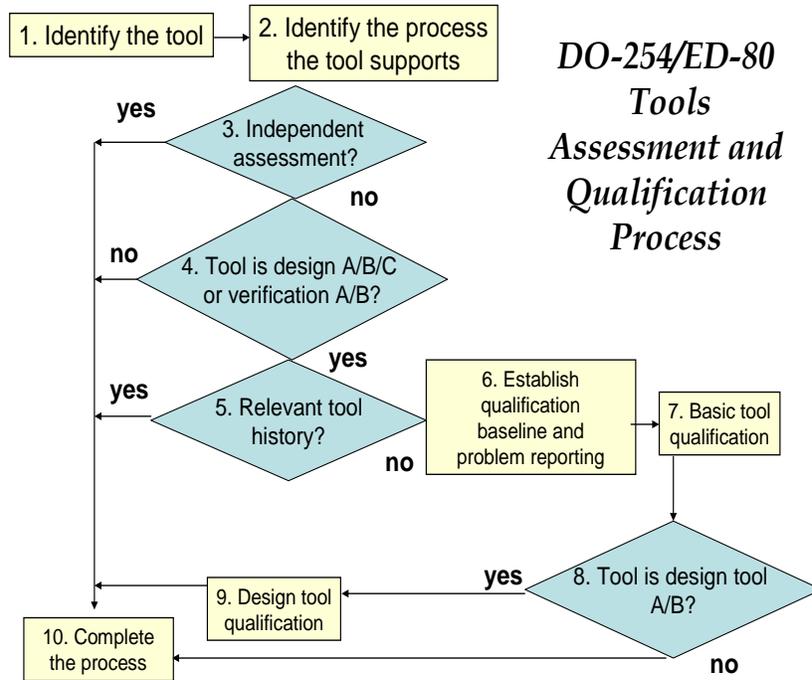


Figure 5. The DO-254 Tool Assessment and Qualification Process [1]

The assessment of the process documented in figure 5 shows four distinct decision elements (highlighted in the figure). Each one of these decision elements offers the opportunity for an independent assessment of the correctness of the design. The correctness of the design will be evaluated numerous ways. The verification suite will assure that the design outputs meet all the logical requirements. Simulations allow the designers to view and analyze the design, using not just the output signals, but every signal in the design. Analysis and debugging of the hardware will allow the designers to verify that the hardware performs as predicted by the design tools. In the typical AEH design, following the verification process shown in figure 3, the process of multiple overlapping independent assessments allows the assertion that (quoting DO-254) “the integrity of the end item does not depend on the design tool output alone.” [1]

In section 4.2, it will be shown that qualification of an FPGA design tool is not sufficient to guarantee correct operation of the hardware, as proving that the design implementation is logically correct does not guarantee correct operation of the hardware. Correct operation of the hardware is heavily dependent on the system environment in which the hardware is operating. Multiple layers of verification will prove more beneficial than attempting to qualify a tool as correct over all voltage, temperature, timing, and environmental conditions.

## 2.11 TOOLS' DISCLAIMERS.

DO-254 indicates that tool qualification is a challenging task, perhaps more difficult than the hardware design itself. The tool vendors understand the difficulty in guaranteeing correct operation over all possible operating conditions. The excerpts presented below are from a manufacturer's product description. Despite great progress and improved trustworthiness of new products, there is no certainty that the product is perfect. This leads to the limited warranty's legal disclaimers, such as:

- Example logic synthesizer:

“<vendor>warrants that the program portion of the SOFTWARE will perform substantially in accordance with the accompanying documentation for a period of 90 days from the date of receipt.

IN NO EVENT SHALL <vendor> OR ITS LICENSORS OR THEIR AGENTS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL OR INCIDENTAL DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTIONS, LOSS OF BUSINESS INFORMATION, OR OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THE SOFTWARE, EVEN IF <vendor> AND/OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES”

- Example simulator:

“<vendor> warrants that during the warranty period Software, when properly installed, will substantially conform to the functional specifications set forth in the applicable user manual.

<vendor> does not warrant that Software will meet your requirements or that operation of Software will be uninterrupted or error free.

<vendor>AND ITS LICENSORS SPECIFICALLY DISCLAIM ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY.”

The simulator vendor does not guarantee that the software will meet the designer's requirements, or that the operation of the software will be error-free. The tool qualification process attempts to provide an assurance that the tool will operate correctly over the operating conditions of interest to the project. Assuring correct operation of the tool requires knowledge of the internal operation of the software. This information is rarely available to the tool user. The tool manufacturers have the necessary information but, because the information often contains trade secrets, they are often unwilling to provide the information required for tool qualification to outsiders.

### 3. ALTERNATIVES TO TOOL ASSESSMENT AND QUALIFICATION.

The purpose of tool assessment and qualification is to ensure that the tool is capable of performing the particular design or verification activity to an acceptable level of confidence for which the tool will be used. In sections 4.2 and 8.1, this research will demonstrate that, even if a design tool's outputs are known to be correct, it is still possible that the implemented hardware will fail to function correctly.

Section 11.4.1 of DO-254 [1] has a comment about level A and B design tools, implying that qualifying a design tool instead of assessing the tool's output, or establishing relevant history, may be more difficult than the hardware design itself.

“Using such a design tool without independent assessment of the tool's output or establishing relevant history is discouraged, as it may prove to be a task as challenging as the development of the hardware for which the tool is proposed to be used.”

DO-254 provides independent assessment of a design tool's output and establishing a relevant tool service history as options in the tool qualification process. In the sections that follow, each method will be examined in detail.

#### 3.1 INDEPENDENT ASSESSMENT OF THE TOOL'S OUTPUTS.

DO-254 states [1]:

“An independent assessment verifies the correctness of the tool output using an independent means. If the tool output is independently assessed, then no further assessment is necessary.”

It is clear that independent assessment of a tool's output avoids the tool qualification process, but what constitutes independent assessment of the tool's outputs?

DO-254 provides the following guidance on independent assessment of a design tool:

“Note: Independent assessment of a design tool's output that is generated in whole or in part by the tool 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 upon the correctness of the design tool output alone.”

Verification activities on the component (hardware) and netlist (software) count as an independent assessment of a tool's output.

For verification tools, DO-254 states:

“Independent assessment of a verification tool's output may include a manual review of the tool outputs or may include a comparison against the outputs of a

separate tool capable of performing the same verification activity as the tool being assessed. The applicant may propose other methods of independent assessment as well.”

Advanced verification techniques, such as elemental analysis or formal methods, can provide independent assessment of a verification tool’s outputs.

### 3.1.1 The Meaning of Independent Assessment.

Independent assessment is more than just the independence of the design and verification tools used. Independent processes are a means to address potential common mode errors that could occur if a single person designs and verifies the hardware item. With a single person performing the design and verification, the verification process will assure that the hardware operates as designed, but this may not be equivalent to how the hardware is required to operate. The responsibility for ensuring that the verification process demonstrates that the design requirements have been met should be performed with an individual, a process, and/or a tool that is independent of the designer. There are many means of establishing the necessary level of independence, and the verification plan should address the specific means to be used for a particular verification activity.

DO-254 enumerates some examples of acceptable means achieving independence [1]:

- “1. Requirements or designs are reviewed by another individual.
2. Test cases or procedures are developed by another individual.
3. Test cases or procedures developed by the designer are reviewed by another individual.
4. An analysis performed by the designer is reviewed by another individual or a review team.
5. A different test is performed that confirms the results of testing by the designer, such as a test during flight test confirms a hardware item test or software verification tests, developed independently and performed on the target hardware item, confirm the results of testing by the designer.
6. Test or analysis results are verified by a tool.”

In addition to design reviews by independent individuals, some automated techniques are allowed. Item 5 allows independently developed tests performed on the hardware to be used to confirm the results of testing by a designer. Item 6 allows other tools (such as formal analysis) to be used to confirm the tool’s outputs.

### 3.1.2 Independent Processes at All Phases of the Design.

It should be noted that rigorous hardware design assurance for dependable systems requires that the major elements of a project—specification (requirements), conception (design), verification (testing), and validation (analysis) must be executed independently by separate groups to avoid perpetuation of errors. In addition, both verification and validation results may need to be reviewed independently to confirm that proper procedures were followed, and that results adequately verify that the requirements have been met. Independence of the design and verification team is a double-edged sword. While it can be a negative for knowledge of the design to influence the work of the verification group, the designers' knowledge of the hardware can help identify areas that warrant particular attention during verification. For designs intended to satisfy DALs A and B, the independence required by the standard must not prevent the transfer of information from the design team to the verification team.

### 3.1.3 What Happens When the Independent Assessment Results Do Not Agree.

Independent assessment data that indicates a failure cannot be treated lightly. While there is always the possibility that the independent assessment is in error, great care must be taken when such a situation is discovered. The mirror on the Hubble space telescope was ground to an incorrect shape, despite the fact that mirror shape was independently assessed with three separate systems. Two of the three measurement systems identified the mirror shape as incorrect. However, the main measurement system was considered to be far more accurate, and the conflicting independent measurements were ignored, eliminating the independent assessment of the mirror shape. It was not until the mirror was in service and determined to be flawed that the error in the primary measurement source was identified [12]. Even in cases where the independent assessment is not as strong as the main design tool, the results of the independent assessment must be carefully considered. It is interesting to note that the losing bidder for the mirror was a team from Kodak and Itek Corporations that proposed to independently build two mirrors and check each other's work. This type of independent assessment would have easily identified the measurement errors.

### 3.1.4 Independent Assessment Is a Process, Not an Event.

Independent assessment of a design tool's outputs should not be viewed as a single event. Independent assessment of the design tool's output is a process that can and should occur continuously throughout the design. There should be multiple independent assessments occurring, each assuring the design correctness at different levels. The independent assessment process follows a bottom-up strategy. The initial independent assessments will use simulations to assure that each signal in each low-level block of the design meets the appropriate hardware and derived requirements. Later assessments will assure that the low-level blocks correctly interoperate as part of the final design. The final assessment during validation will assure that the design meets all system-level requirements in the actual system application. The independent assessment process operates the same for all DALs. Level A designs will require far more extensive assessment than level C designs in order to assess the design robustness and any architectural failure mitigation techniques that may be in place.

It is often possible to assure some lower level requirements in later assessments, such as at the system level, but many low-level requirements cannot be assured at the system level. It is also possible that there are requirements that can only be fully assessed at the system level. In this case, the assessment at lower levels should cover as much of the requirement as possible. At every stage in the assessment process, the assessment should attempt to verify as many design requirements as possible and to assess each requirement to the fullest extent possible.

### 3.2 SERVICE HISTORY.

In the absence of an independent assessment of a tool's output, the DO-254 tool assessment and qualification process allows a tool's relevant service history to be used as an alternative to tool qualification. In section 11.4.1, DO-254 [1] gives the following guidance with respect to tool service history:

“The history of the tool may be based on either an airborne or nonairborne application, provided that data is available to substantiate the relevance and credibility of the tool's history.”

Design and verification tools from mainstream vendors, such as Mentor or Synplicity, have been used by hundreds of users on thousands of predominantly nonairborne projects. Mainstream vendors actively track problem reports and keep change logs for the software revisions. Many of the projects are more complex and speed-intensive than is typical for aviation applications. But very few of the nonairborne projects are concerned with safety-critical hardware; therefore, the relevance and credibility of the tool history should be questioned.

#### 3.2.1 Service History Case Studies.

A good service history does not assure that a design is error-free. Consider the following examples, in which significant system failures occurred despite an excellent service history:

- The Intel<sup>®</sup> floating point processor circuitry was incorporated in both the 486<sup>™</sup> and Pentium<sup>®</sup> processors. There were hundreds of thousands of these processors in use for several years without any issues or failures related to the floating point processor. In 1994, a user identified an error in the floating point circuitry. Byte Magazine estimated that the error would occur once in every 9 billion randomly generated calculations [13].
- The European space agency spent 10 years and \$7 billion to produce the Ariane 5 rocket. In 1996, at 36.7 seconds into the maiden flight, the inertial navigation system attempted a data format conversion and ran into a number that was too large for the destination register to store. The resulting error caused the navigation system to shut down. The backup navigation unit came online and ran into the same problem. The rocket lost navigation control and was subsequently destroyed in flight [14].

To apply the guidance of DO-254 to the above cases, the task begins with assessing the relevance of previous applications, installations, and environments to the target application. The Intel

floating point processor was used in numerous previous processor designs. The application, installation, and environment of the new processor applications were identical to previous successful applications. The application of the navigation system of the Ariane 5 rocket was identical to the Ariane 4. The physical installation was identical, except for differences in the dimensions of the equipment bay. The environment appeared identical; it connected to similar hardware, on a similar rocket. Engineering analysis of both the Intel and Ariane examples concluded that the new applications were identical in application, installation, and environment. Regarding the actual failure rates in operation, there were no known functional failures of the floating point processor or the Ariane navigation system. Both systems had long and problem-free service histories.

Both systems would meet all the criteria necessary to qualify based on service history. The service history did not predict the future performance because the bug in the floating point processor had slipped by detailed verification by Intel. It was also sufficiently rare that randomly generated test patterns had a negligible probability of finding the bug. If 100,000 users in the user base averaged a single floating point calculation per second, the error would have occurred at a minimum of once a day for years. The actual rate of occurrence is expected to be substantially more frequent. Despite the high error rate, the error was not detected because the error was not producing detectable errors in user applications. When an application requiring extreme precision independently assessed the calculations, the error was identified.

The excellent service history of the floating point processor was based on users doing general computing. This proved a poor predictor of the system performance for an extreme precision application. The excellent service history caused even the mathematician who discovered the error to suspect that the problem was in his calculations, and not in the hardware.

In the case of the Ariane 5 rocket, the particular subsystem that failed was only used to align the navigation system before launch. The system that generated the failure could have and should have been turned off prior to launch. However, in a decision made many years earlier, the navigation system was left enabled for the first 40 seconds of flight to make it easier to restart the system if there were holds on the launch pad. The navigation system failure occurred because the Ariane 5 was a much faster rocket and generated velocities that could not be achieved on the Ariane 4.

The new application of the navigation system was identical in all physical respects, but the data coming from other systems had changed. Again, service history yielded a poor prediction of future performance because the environment the system operated in had changed in a subtle way that had not been recognized.

### 3.2.2 Service History Guidance for Hardware.

DO-254[1] gives guidance about product service histories for hardware in Section 11.3 and Appendix B, Section 3.2. Sections 11.3.1 and 11.3.2 of DO-254 give guidance on product service experience acceptability. One or more of the several criteria should be met to determine the product service history acceptability. Quoting DO-254, these criteria include:

“Assess the relevance of previous applications, installations and environments to the target application, based upon engineering analysis.”

and

“Actual failure rates in operation.”

The guidance in Section 3.2 of appendix B of DO-254 [1] requires additional design assurance for DAL A and B hardware if service experience is claimed. It also contains a requirement to link any analysis of product service history experience into the FPPA for levels A and B.

### 3.2.3 Service History for Design Tools.

The performance of hardware is the same regardless of who is using it. If the application, installation, and environment are similar for multiple applications, it is reasonable to expect that, if the hardware worked well in one application, it will work well in the other applications. As shown in appendix F, this does not extend to design and verification tools. The performance of a design or verification tool instead depends heavily on the abilities and experience of the tool user. Experienced users can work around the known weaknesses and bugs of a known design tool. The user’s experience with the tool is a better predictor of design correctness than the tool’s service history.

### 3.2.4 Service History vs. the Latest Technology.

When it comes to safety, there can be a trade-off between service history and new technology. Consider the introduction of airbags in the auto industry. The airbag was first offered as optional equipment in passenger cars in 1975. The first recorded accident between two vehicles in which an airbag deployed to protect each driver occurred in 1990 [15]. It took until 1998 before the government made airbags mandatory equipment for all passenger vehicles. Part of the reason for this delay was that legislators wanted a known service history and safety record before airbags were mandated. Many lives could have been saved and highway safety improved if the government had made the new technology mandatory sooner.

Design and verification tool technology is constantly improving. New tools may produce far safer designs than older tools with documented service histories. Independent assessment of the tool’s output provides far more assurance of design correctness than tool service history.

### 3.2.5 Tool Service History Is Not Sufficient.

It does not appear that a relevant service history alone is sufficient to prevent errors in the final design. The authors recommend that the design tool’s service history should not be allowed to avoid tool qualification. The tool’s service history should be used instead to modulate the level of independent assessment effort required. A tool that lacks a relevant service history should be subjected to additional independent assessments to assess the correctness of the tool.

Since there will be other independent assessments of the tool's outputs, these additional assessments need not be as exhaustive as those required for tool qualification. The additional assessment can be an independent manual review of the outputs of the new tool, analyzing the design with both the new and old tools and comparing the results. In some cases, a tool with no relevant service history may address an issue that was not addressed by any previous tool. In this case, the correctness tool outputs should be subjected to a manual review.

### 3.2.6 Testing Maturity Model.

The Testing Maturity Model (TMM), introduced in 1996, did not find much acceptance because of its limited documentation and theoretical style [16 and 17]. However, the approach is still promoted by the TMMi Foundation (<http://www.tmmi.org>). TMM focuses on testing inspired by the Software Engineering Institute Capability Maturity Model, which assumes that there is a correlation between organizational maturity and the quality of produced software. Because the TMM approach is primarily used in enterprise computing and information technology organizations and, because the testing component of verification and validation is adequately addressed in the existing guidance, the TMM approach has not been considered relevant for AEH.

## 4. DESIGN ASSURANCE.

DO-254 defines design assurance as follows [1]:

“Design Assurance—All of those planned and systematic actions used to substantiate, at an adequate level of confidence, that design errors have been identified and corrected such that the hardware satisfies the application certification basis.”

The key to design assurance using DO-254 is identifying all possible design errors. Before a design error can be identified, a test case that produces the error must be generated, and then the error must manifest itself in a way that the test system can detect. The following sections investigate methods to generate more complete test cases, and to better observe errors when they occur.

### 4.1 CONSTRAINED RANDOM VERIFICATION.

Constrained random verification is a technique for which the input conditions of the device being tested are bounded according to the hardware requirements. Within these bounds, test cases are randomly constructed and used to assess the correctness of the design. If a condition is not prohibited by the requirements, then it can be used as a test case.

Because the test cases are computer generated, they often identify failures that are legal conditions that were not considered by the design and verification teams. Because the input constraints are defined by the hardware requirements, constrained random verification often identifies problems with the hardware requirements. As part of a normal design process, many

semiconductor companies run a new set of random test cases on the design in progress every night.

Synopsys published a study on a USB 2.0 Host Controller IP core [18]. The core had been verified by traditional simulation methods, as well as manual-directed tests backed up by some random testing in Verilog. The suite consisted of 450 directed/random tests and achieved what seemed to be excellent coverage results:

- 97.50% finite state machine (FSM) coverage
- 88.64% toggle coverage
- 84.71% condition coverage
- 98.58% line statement coverage

Given a design that was well verified in simulation, hardware verification was then performed on the design. A total of 25 new bugs were found. The design bugs were put into four classifications:

- B4—show-stopper bug that could prevent a product from working
- B3—significant functional bug that would affect some applications
- B2 and B1—relatively minor bugs, usually with workarounds

The 25 bugs discovered during hardware verification were in the B3 and B2 categories. There were no show-stopper bugs found, so the initial verification effort was successful. However, the hardware verification still found and fixed a number of important problems. At this point, there should be enough evidence that the design has been assured to be correct.

Constrained random verification was then performed on the already well-verified design. An additional 28 bugs were found:

- 15 were B3 level
- 12 were B2 level
- 1 was B1 level

Constrained random verification helps generate test cases to identify bugs that slip by the normal design and verification efforts. Constrained random verification should be part of any safety-critical AEH design flow.

## 4.2 OBSERVABILITY.

For safety-critical designs, both the verification and validation test suites need to be assessed for completeness. Elemental analysis is often used to assure that every element of the design is exercised by the test suite. It is important to realize that merely exercising every element is not sufficient to assure the design is correct. If an element makes an error, it is necessary for the error to propagate to an observable point for the error to be detected. Consider the case of several circuits connected to a single output. If an error in one circuit occurs while another

circuit has control of the output, the error will go undetected, but the elemental analysis can report that the element was tested and the error may or may not have been reported.

The number of internal states usually far exceeds the number of outputs, so visibility into the circuit's operation is limited. Modern design methodologies use assertions to give better visibility into internal circuit operation during simulation. An assertion acts like a comment that checks itself during simulation. Suppose that, as part of his HDL design process, a designer puts in assertions, such as the contents of a four-bit counter must always be between 0 and 10. If during any simulation the counter contains an illegal value, the assertion fires and the violation is noted in the error log. Even if this condition quickly goes away and the illegal value does not propagate to any observable points, it still generates a detectable error.

If the hardware fails during system validation, determining the failure mechanism in the hardware often requires that the conditions leading to the failure are simulated. The assertions will provide a detailed map of where and when the failure originated and where and how it propagated. Assertions are widely used in the semiconductor industry because of their ability to improve the observability of failures and their usefulness as a debug tool if the hardware fails in the field or during system validation. An additional benefit of using assertions is that they highlight how well a design component is specified.

#### 4.3 DERIVED REQUIREMENTS.

DO-254 defines a derived requirement as follows:

“Derived Requirement—Additional requirement resulting from the hardware design processes, which may not be directly traceable to higher level requirements.”

Derived requirements that impact safety must be verified. DO-254 lists some example conditions in which derived requirements may address safety conditions:

“Note: Derived requirements may address conditions, such as:

- a. Specific constraints to ensure that functions of a higher design assurance level can withstand anomalies of functions of a lower design assurance level as seen at the interface of the function with the lower design assurance level.
- b. The range of data inputs considering typical and full-scale data values as well as the high and low states of bits in data words or control registers.
- c. Power-up reset or other reset states.
- d. Supply voltage and current demands.
- e. Performance of time-related functions, such as filters, integrators and delays.

- f. State machine transitions that are possible, whether they are anticipated or not.
- g. Signal timing relationships or electrical conditions under normal and worst-case conditions.
- h. Signal noise and cross-talk.
- i. Signal glitches in asynchronous logic circuits.
- j. Specific constraints to control unused functions.”

Assertions cannot cover timing, power, or noise issues (conditions d, e, and h), but all other derived requirements can and should be covered by one or more assertions. Covering a requirement with an assertion provides an independent assessment of the design tools output.

## 5. SURVEY OF TOOL USERS.

To identify issues and concerns in AEH tool qualification and certification, one must start with a broader view of an industry perspective. This section reports on the survey of the aviation community conducted to collect relevant information.

### 5.1 AVIATION COMMUNITY SURVEY.

The survey was conducted to collect data on experiences and opinions concerning the use of programmable logic tools as applied to the design or verification of AEH (FPGA, PAL, GAL, PLA, ASIC, or SoC) according to the DO-254 standard. The questionnaire was sent out, targeted toward individuals who have experience with developing or using such tools or experience with qualifying such tools. The purpose was to gather industry and certifying authority feedback on assessment and qualification of AEH programmable logic tools.

#### 5.1.1 Survey Population.

The questionnaire was distributed first during the 2007 FAA National Software and Airborne Electronic Hardware Standardization Conference in New Orleans, LA, July 24–26, 2007, which was attended by over 200 participants. A special session dedicated to AEH was attended by 54 individuals, representing industry and government organizations interested in AEH and the application of DO-254. In addition to distributing and collecting paper copies of the questionnaire at the conference, a follow-up mailing was distributed to over 150 individuals engaged in the development of aviation software and hardware. The questionnaire was also distributed internally within several companies engaged in the design of PLDs. As a result of these activities, a sample of only 17 fully completed responses was received. As a follow up, surveys were distributed at the Programmable Logic User Group meeting in Clearwater, Florida on November 15, 2007, resulting in three more responses. This is a rather disappointing outcome and a potential risk issue. However, the collected results provided several interesting observations.

In January 2008, using an external survey website (www.surveymonkey.com), the questionnaire was posted on the web and followed up with an additional 266 mailings requesting response. Additionally, the link to the web survey was placed on the DO-254 Users Group website (select the “tools” tab). Only eight responses were received.

A copy of this questionnaire is included in appendix A. The detailed results and the majority of relevant graphs are presented in appendix B. The general conclusions of the survey, based on the current respondents database, are presented below.

Figure 6 shows the survey population by organization type. The majority of respondents work for avionics or engine control developers (~65%). Over 95% of the respondents have a technical background, with ~55% having bachelor’s degrees, ~45% master’s degrees, and over 72% having an educational background in electronics. Ninety-seven percent of the respondents have more than 3 years of experience, with fifty-nine percent having more than 12 years of experience.

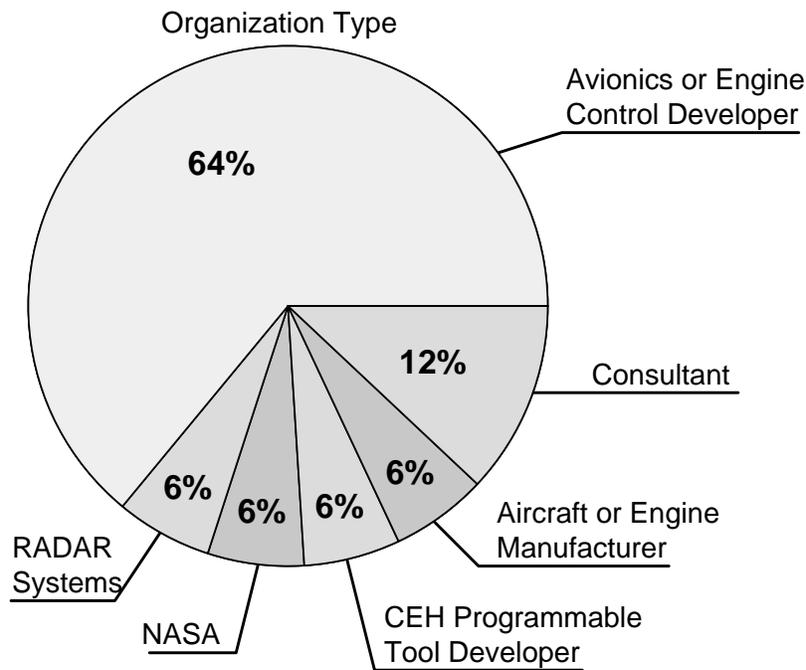


Figure 6. Survey Population—Type of Organization

### 5.1.2 Multiple Choice Answers.

The most frequent roles of respondents relevant to the AEH tools were as follows (see figure 7):

- Use of the tools, including development/verification of systems (~62%)
- Managing and acting as DER (~26%)
- Development of the tools (~2%)
- Development of components (~12%)

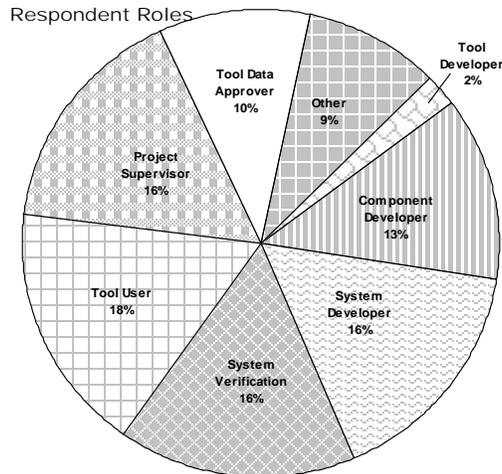


Figure 7. Role of Respondents in DO-254 Projects

The respondents' primary interest was divided between verification (32%), development (27%), hardware (22%), and concept/architecture (18%).

A wide range of devices were used, with the most frequently used being FPGA (~27%), CPLD (~18%), ASIC (~15%), PAL (~11%), PLA (~9%), and Erasable PLD (~8%). The most popular hardware device vendors are Actel (~27%), Xilinx (~24%), Lattice (~13%), and Cypress® (~11%), with Quick Logic®, Altera, and Atmel® below 10%.

The most widely used tools are from Mentor Graphics (~27%) and Synplify® (~22%), followed by Synopsys (~17%), Aldec™ (~11%), and Cadence (~8%). About 23% of respondents use other tools.

In regard to criteria for the selection of tools for use in DO-254 projects, the most important are availability of documentation, ease of qualification, previous tool use, and host platform, followed by the quality of support, tool functionality, tool vendor reputation, and previous use on the airborne project. Selection of a tool for a project is based either on a limited familiarization with the demo version (50%) or on an extensive review and test (40%). The approach of reviewing and testing the tool by training personnel and using the trial period on a smaller project seems to be prevalent.

For those who have experienced the effort of qualifying programmable logic tools (only 14% of respondents), the quality of the guidelines was considered sufficient or appropriate (62%), as is the ease of finding required information (67%), while the increase in workload was deemed negligible or moderate (80%). An interesting observation concerns the scale of safety improvement: marginal (43%), moderate (21%), noticeable (7%), and significant (29%). Similarly, the question about errors found in the tools may be a source for concern: no errors (11%), few and minor errors (50%), and significant and numerous (17%). Despite all this, the satisfaction level regarding programmable logic tools was positive, and more than 96% of the respondents marked their satisfaction level 4 out of 5.

Complete results in a graphical form are included in appendix B.

### 5.1.3 Narrative Answers.

Three questions from the questionnaire required some form of narrative answers. The transcript of the unedited answers is in appendix B following the statistical data from the survey.

In summary, the following were identified as major issues for using programmable logic tools:

- Ease of using the tool for verification and reading the results of the tool
- Quality of the tool
- Speed with which the tool verifies designs
- Problems with timing and timing analysis

Difficulties identified with qualification of these tools include:

- Lack of useful guidance for how to qualify tools
- Lack of cooperation from the tool vendors
- Frequent version updates of tools
- Length of qualification process

Other issues identified include:

- Difficulties compensating for certain tool idiosyncrasies
- Lack of configuration control
- Lack of other vendor cooperation

## 5.2 SEMICONDUCTOR INDUSTRY VIEWPOINT.

While working on the results of this survey, a similar survey targeting hardware development tools without a safety objective was discovered [19]. Although the population surveyed was primarily composed of chip and low-level component designers, the results show general industry trends and viewpoints regarding the use of tools. This particular survey was done in consideration of the fact that the IC and digital components industry uses the tools daily. There is a significant number of digital system developers who are unfamiliar with the specifics of DO-254.

The Census was sent to 25,000 members of the E-Mail Synopsys Users Group and 818 responses were received. The objective of the survey was to identify the actual utility of the tools, as opposed to the market share expressed in dollars, which is the statistic often presented in trade literature.

The major items identified by the survey were:

- Specific language use for simulations: Verilog<sup>®</sup> was used exclusively or primarily at 73% and VHDL at 20%. It was observed that the justification for using Mentor's ModelSim<sup>®</sup> was the need to provide support for legacy code and reuse.
- The most popular simulators are Synopsys VCS<sup>™</sup> (44.7%), Mentor ModelSim (35.3%), Cadence NC-Sim (24.3%), and NC-Verilog<sup>®</sup> (18%).
- The most popular waveform/debug tools include Synopsys (33.2%), Novas<sup>™</sup> Debussy<sup>®</sup> (33.1%), Cadence debugger (29.6%), and Mentor MTI debugger (26.3%)
- Only 23% of respondents use SystemC, mostly for high-level modeling and verification; the most popular SystemC tools include Free OSCI (43%), Cadence MC-System (33.6%), and Mentor ModelSim (16.8%).
- 35.1% of the projects use System Verilog, almost exclusively, for testbench (80.2%); Synopsys VCS is the most popular (65.6%), with Cadence NC-Sim (24.7%), Mentor Quest (15%), and Mentor ModelSim (12.3%) following.
- The assertions are used by both designers and verification personnel and the responders found their application useful (89.4%); the most popular was System Verilog SVA (37.8%).
- Use of formal "bug-hunters" was not popular (74.5% respondents do not use them); Mentor 0-In, Synopsys Magellan<sup>™</sup>, and Cadence ISV/IFV/BlackTie were most often quoted.

It should be noted that most of the companies use more than one tool from multiple vendors. Such an approach introduces an additional element of independence, assuring that products of different vendors are applied to design and verification, thus avoiding potential exacerbation of errors.

As part of the research into design and verification tool use by AEH developers, the Vice President of Design Services of a two billion dollar company was interviewed. He had a number of interesting observations relative to the research. His company used Cadence tools for both design and verification. When asked if he felt that there was any risk in using design and verification tools from the same vendor, he replied that even within a single vendor the design and verification tools are very different and independently developed. His primary concern was using the tool that performed the best in his applications, rather than having independent vendors for the design and verification tools.

He mentioned that the company uses formal design tools for smaller blocks and at an architectural level, but they did not find them suitable for complete design verification. He described his verification flow as:

1. Assertions are incorporated as the HDL code is written.
2. Formal proofs are performed on small blocks as soon as possible.
3. Directed tests are generated by the designers to exercise areas of concern and constrained random tests are run on a daily basis as the design progresses.

It is noteworthy that the designers have direct input into the verification process to assure that areas that the designers viewed as areas of concern are adequately verified.

He also mentioned that newer verification management tools were extremely useful in helping to evaluate the verification coverage of the entire system, from low-level blocks to system-level coverage. These tools help guarantee that all the blocks in a design have been correctly exercised with directed tests, constrained random testing, fault coverage, and assertion coverage.

## 6. LITERATURE OVERVIEW.

One of the project objectives was to research the literature related to the use of software tools for design and verification of AEH. Literature research was categorized from three perspectives:

- A general research perspective, which gives a broad view of the issues involved in designing AEH.
- A focus on safety issues in avionics applications, which discuss more specific problems related to safety-critical aspects of AEH development.
- An industry perspective, which provides the most detailed view of industry practices in qualification of AEH tools with respect to their compliance with DO-254 standard.

An attempt was made to collect the literature providing a general overview of the research issues related to the use of software tools in the development of AEH. These papers are presented in appendix D, with the respective entries grouped in section D.1 (papers 1-26). Each entry includes an abstract of the respective paper, which is listed in references 20-45.

Six of the selected papers contain explicit relevance to safety-critical aspects in avionics (papers 12, 13, 17, 22, 24, and 25). This discussion is elaborated on in section D-2 in appendix D. The objective of this discussion is to provide synopses of selected research papers that are directly related to issues raised with respect to safety concerns in the area of AEH with a focus on avionics applications. The purpose of this analysis is to identify the issues raised by researchers and developers concerning the use of AEH tools for design and verification of PLDs. The specific concerns are as follows:

- Input/output (I/O), (to determine the state of undefined I/O pins)
- Power (routing connections within an FPGA so that electro-magnetic fields and maximum current draws do not affect the logic or output voltage levels)
- Simulation (high-level behavior simulation and implementation behavior are not always identical)
- Timing (the tool meets the timing constraints that were displayed in its report and it retains a margin of safety)

Each paper has been analyzed to identify the objective, brief description, and the relevance to the project.

The papers reviewed in the safety category suggest the following actions should be taken in regard to the use of PLDs in advanced airborne applications:

- Plan to develop and verify PLD programs in the same way as software programs.
- Plan the safety argument from the start and build up evidence throughout development.
- Use mature tools, amenable to qualification and supported throughout the project life time.
- Investigate the use of formal notations and analysis techniques to increase verifiability.
- Do not use programmable logic hardware just to avoid developing safety-critical software.

The third perspective of the literature research focused on industry practices related to AEH tools qualifications, as reported in related articles. These entries are grouped in section D.3 of appendix D. Each entry includes a brief description of the problem discussed in the paper and a suggested solution. These papers are also listed in references 46-73.

Ten of these papers deal directly with the vendors' views on tool qualification according to DO-254 [1]. Mentor Graphics [46] and TNI-Software [59 and 63] describe their approach to comply with DO-254 for their respective verification tools: ModelSim, Reqtify<sup>®</sup>, and a formal property checker, imPROVE-HDL. Two COTS tools from the GNU package, a configuration management tool, Concurrent Version System (CVS), and a problem-reporting tool, GNATS, are recommended in reference 51. Four vendors, Xilinx, Altera, TNI-Software, and Mentor Graphics, identify their tools and processes in reference 55, and Aldec and Barco-Siles S.A. outline their processes to comply with DO-254 in references 60 through 62, respectively. Airbus [48] and the DO-254 User Group [54] outline their processes separately, with a list of issues and clarifications regarding compliance.

Four papers take a guideline approach to clarify respective AEH issues, whether related to DO-254 or not:

- The objective of the National Aeronautics and Space Administration (NASA) [47] is to provide guidelines to improve understanding of AEH among those interested.
- The FAA report on COTS [52] identifies key attributes to meet the DO-254 objectives, but falls short of relating them to tool qualification.
- The ERA Technology, Ltd. [53] covers DO-254, but does not address the tool qualification issue.
- Reference 74 discusses DO-254 briefly and identifies the Avionics Process Management Committee's EIA-933 Standard providing recommendations on how to select and manage suppliers of avionic products.

The remaining four papers discussed in this section present academic and research views on the issue of certification. Lundquist [56] addresses the question of certification of an Actel FPGS chip and concludes that this question "remains unanswered." Hilton and Hill [49] advocate the use of Synchronous Receptive Process Theory to reason about the FPGA as a collection of small processes reacting to signal inputs. Jacklin et al. [57], argues that complete verification and validation of learning systems should not be viewed as running test cases and comparing expected results to actual results because these tests can never reveal the absence of errors. Finally, Crum et al. [58] point out that the lack of research investment in certification technologies will have a significant impact on levels of autonomous control approaches that can be properly flight certified and could lead to limiting capability for future autonomous systems.

Additionally, the research explored a variety of web references and collected nearly 300 positions from SOCCentral, a webpage containing multiple articles related to ASIC, FPGA, electronic design automation (EDA) and IP.

## 7. CASE STUDIES.

There are a number of factors that may affect system safety, such as system quality, complexity, user experience, fault tolerance, producer's pedigree, documentation, testing, tool quality, and quality assurance (QA). These factors are shown in figure 8.

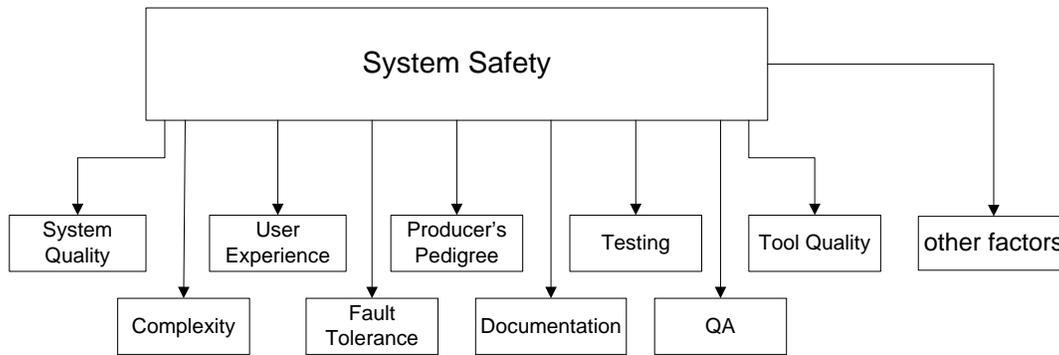


Figure 8. Factors Affecting System Safety

To evaluate tool quality completely, one would need to look at it from three different perspectives and collect the following data accordingly:

- How the tool itself was developed.
- How the tool is operating.
- How the quality of the product developed is affected by the use of the tool.

The framework for this process, based on the context of tool use, is shown in figure 9 and is taken from a paper by Kornecki and Zalewski on tool evaluation [75] that states:

“The central part of this model is the *macroevaluation* based on the use of the tool during the design phase. However, much information on tool quality can be derived from the development of the tool itself, considered as a *metaevaluation*: evaluating the process to develop a tool. The tool vendor can provide the data for evaluation of this stage. In addition to the *macro-* and *metaevaluation*, the product developed with a particular tool can be included in the evaluation. This is called *microevaluation*, and it focuses on the level lower than the tool itself. Such a product evaluation can be based both on static code analysis and code execution. Consequently, to have the entire picture of the tool’s quality, one needs to do the evaluation at three different levels.”

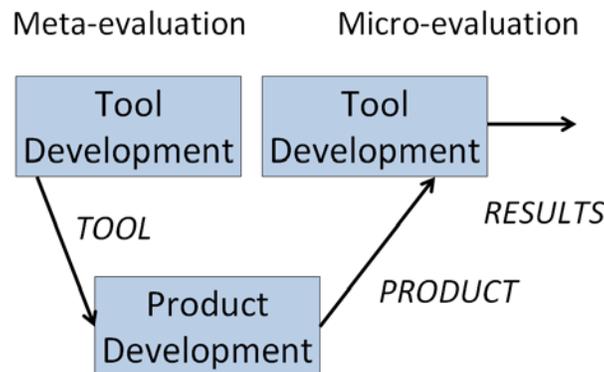


Figure 9. Macroevaluation Model of the Tool Evaluation Process

It is next to impossible to obtain data on tool development from the tool vendors. This is mainly due to the vendor's reluctance to release proprietary information to the public where it could be used by competitors. For this reason, performing the meta-evaluation is normally not done. Therefore, this work focuses on macroevaluation, described in appendix F, and microevaluation, described in appendix E.

## 8. SAFETY ISSUES.

This section focuses on safety issues that can occur in a design using tools that have been through the full qualification process. Using these tools, it is possible to produce systems that have been proven to operate correctly under all anticipated conditions. However, this confidence is based on the assumption that the system and all of its related systems will operate as expected. The examples in this section emphasize the need for independent assessment of a tool's output. This section will begin with examining the data from the semiconductor industry on the types of design errors prevalent in hardware designs. Each of these errors will then be examined in more detail.

### 8.1 HARDWARE DESIGN ERROR CHARACTERIZATION.

Figure 10 shows data on designs that required two or more re-spins to get correct. In a semiconductor design, the entire design is fabricated on a single piece of silicon. An error anywhere in the design requires new silicon to be fabricated using new mask sets. This is known as re-spin, and its cost can exceed \$1 million. The data from a 2-year market study (2002 and 2004) categorizes the number and types of errors found in semiconductor designs. The defects total to more than 100% because a single design may have several types of errors. The most common error is the logical/functional error. This error is often caused by inadequate specification of the design. Other errors of interest are delays, clocking, and fast path and slow path errors, all of which are timing-related errors. The number of all timing-related errors exceeds the number of logical and functional errors. IR (voltage) drops is a shorthand way of describing voltage drops due to resistance in the power and ground supply networks. Glitches are unexpected signal transitions that can be timing or signal integrity-related problems. The chart shows that there are numerous ways that a design can be functionally correct but still produce errors in operation.

## Functional Flaws Driving Need for Re-Spin

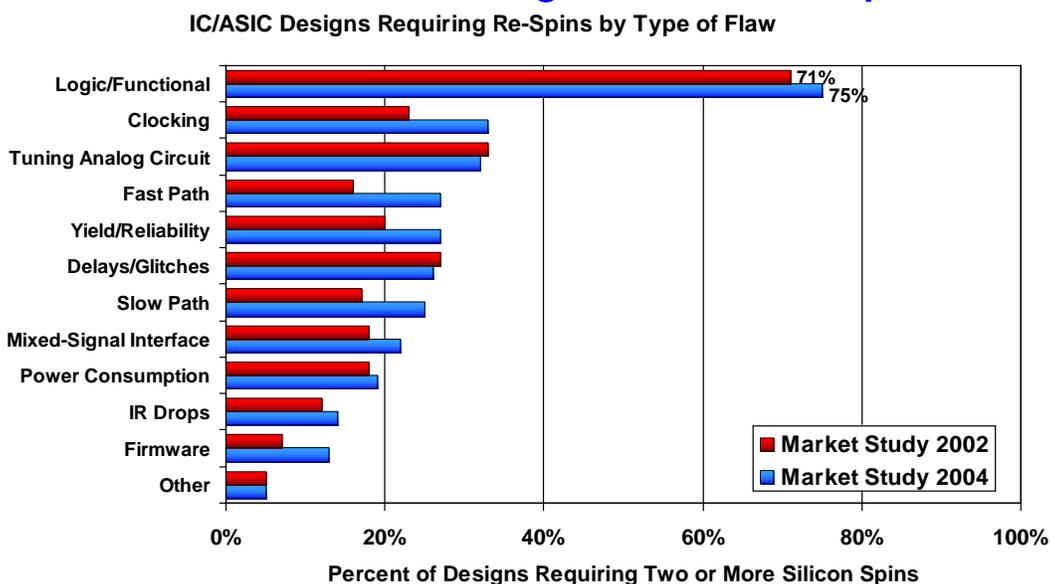


Figure 10. Functional Flaws Requiring Design Re-Spins [66]

In 1965, Gordon Moore postulated that the number of transistors on an IC will double every 2 years. This law has held true for over 40 years, and it appears that it will continue to hold true in the future. Therefore, when comparing the data of figure 10, one must understand that, on average, the designs of 2004 contain twice as many transistors as the designs of 2002. Given that the designs produced in 2004 are roughly twice as complex as the designs produced in 2002, one would expect that the number of category errors would nearly double as well. However, the number errors did not double—because the designs produced in 2004 were more advanced than in 2002.

The foregoing example indicates that if one wants to minimize the number of design errors, one should use the latest tools and design techniques. This example is counter to the idea of requiring an established tool service history.

A closer examination of the year-to-year data shows that, as the number of transistors (or complexity) increases, the number of timing-related errors rapidly increases relative to the number of logical errors. This indicates that tool assessment needs to not only assess whether a tool's outputs are logically correct, but also the accuracy of the tool with respect to timing analysis. Assessing the correctness of the timing analysis is a difficult problem because identical semiconductor devices will not have identical timings. The need to accommodate device variations, temperature variations, and supply voltage variations results in broad timing specifications that make it difficult to assess the correctness of any single timing calculation. It is difficult (if not impossible) to assess the timing accuracy of a tool without a defining architecture and physical layout. The timing calculations of a tool can only be assessed within the context of a design.

Generating functionally correct HDL code is only one part of the overall FPGA design process. Equally as important as HDL code functionality is the location of the HDL code implementation. Locating all the HDL blocks close together improves the timing at the risk of introducing power distribution problems. However, spreading the blocks apart may introduce clock skew and timing problems. In addition to the locations of the individual blocks, the actual location of every input and output pin that is used must be defined. The location of these pins is defined by interface requirements (pins with similar power requirements are grouped together) and external system requirements. The FPGA ultimately connects to the rest of the system through the printed circuit board (PCB) on which it is mounted. Because of the flexibility of the FPGA internal routing, the FPGA pin-out is often determined by the routing constraints of the PCB. Although the FPGA designer may know that clock inputs should not be placed near large-output buses, the physical layout of the PCB may force a suboptimal pin-out upon the designer. Figure 11 shows the synchronization and interactions between processes involved in putting a programmable IC, such as an FPGA, on a PCB.

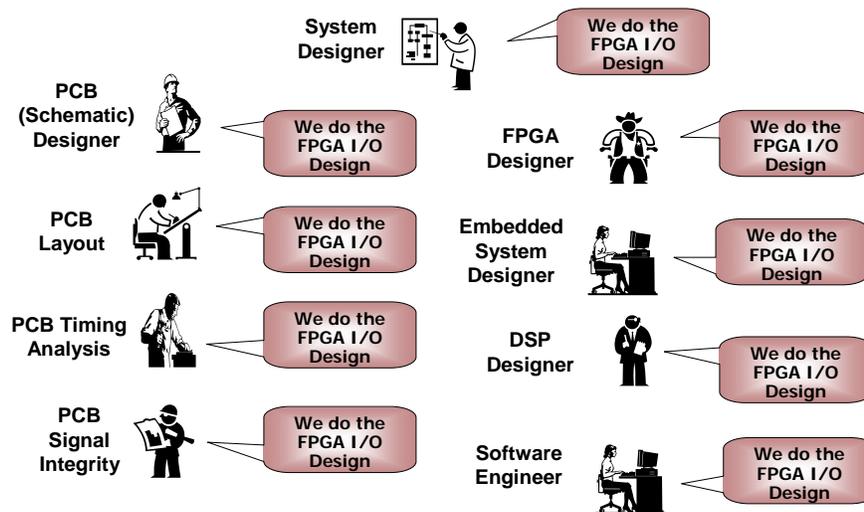


Figure 11. Communication Barriers That Can Prevent Clear Design Specifications [76]

The FPGA is usually part of a larger system; therefore, successful design requires the close collaboration of a variety of specialists. Figure 11 shows ten domains of the engineers working on a specific project that all have a part in specifying the I/O interface. Lack of communication between any of these members can cause incompatibilities in the design, which can lead to failures. For example, improper power distribution in the PCB could cause the FPGA to malfunction, or signal integrity issues in the PCB can allow signals to interfere with each other and produce failures. Therefore, the designer must understand the relation between the system, algorithms, board-level circuitry, PLD, and microprocessor software.

In the sections that follow, assume that a DAL A design that has been proven to be logically correct and the techniques of appendix B of DO-254 [1], including architectural mitigation and elemental analysis, have been applied to the design. Is it possible for the design to not operate as expected in the final system? The following sections investigate issues ranging from PCB design to neutron radiation effects.

## 8.2 THE FPGA'S ENVIRONMENT.

The FPGA is mounted on a PCB that supplies the device with power and routes the signals to the necessary locations. Errors in the PCB design can lead to power integrity or signal integrity problems. These problems are difficult to identify because they heavily depend on the particular operational mode that is implemented in the FPGA. Signal rise and fall times generated by modern FPGAs continue to become faster. This means that signals can no longer be arbitrarily wired together. High-speed signaling requires design engineers with experience in high-frequency analog design to assure that the traces are correctly matched and terminated on the PCB. In older systems, the signal rise and fall times were often slow enough that signal integrity issues did not matter. Many designers currently in the field are unaware of the need to use high-speed signal design techniques. Designers (especially the most senior designers) often do not realize that the rise and fall times of the signals they are now using require a new design paradigm. If the engineers are unaware of the potential failure, the processes of DO-254 and its appendices will not help to prevent signal integrity-related failures.

Wide data buses can cause large supply currents that can interact with the inductances in the PCB leading to supply variations at the PCB level. Capacitors on the PCB can interact with parasitic inductances to produce unexpected resonances. These failures are known as power integrity failures. The increased switching speeds of modern FPGAs place much more stringent requirements on the system power supply design than were placed in the past.

In system designs, it is good design practice to reuse previous successful designs. However, one must be aware that a design that worked fine for one FPGA implementation may fail if the circuitry implemented in the FPGA or the devices connected to the FPGA change. Because they are data- and timing-dependent effects, signal integrity and power supply integrity issues are difficult to identify during the design or in simulation. Best practices design techniques can be used to identify and address problem areas in the design phase. Any errors of this type that slip through should be observable during system validation, if the validation testing includes tests that are intended to exacerbate signal and power integrity issues. The FPGA can only be assured to work correctly if the PCB is attached correctly. Assuring the PCB design correctness does not directly fall under DO-254 scrutiny and often requires a completely different skill set than assuring the correctness of the FPGA.

## 8.3 TIMING ISSUES.

Perhaps the most difficult aspect in verifying the correctness of hardware is that minor changes in the timing can produce major differences in the logical operation of a circuit. To better understand this problem, consider a bus of many bits that instantaneously transitions from all of the bits being zero (0) to all of the bits being one (1). Due to differences in the routing and random variations in the devices, some bits will transition faster than others. This results in a period of time where some of the bits are stable and some of the bits are still transitioning. During this period, the data on the bus is invalid. Accurate simulation of this timing variation requires knowledge of the exact placement and routing of the devices. Any simulation not incorporating timing data from the place-and-route process will not be able to see these differences. In addition, if the simulation timing step size is larger than the timing differences,

then the timing differences will not show up in the simulation output. One must always be aware of the limits of the simulation.

It is possible to minimize the timing variations caused by routing differences by placing timing constraints on the design tools. However, there is always a timing variation due to random device variations; these effects are rarely (if ever) included in logic simulator models. A 2005 presentation [67] indicated that there was a problem with the VHDL design tools because the output signal simulations of the Gray code circuit were not valid at all times. The author does not mention trying to eliminate the timing difference by constraining the design, but instead shows that changing the VHDL code eliminates the glitches in the simulations. What the author failed to grasp is that even when the simulations show that the data is always valid, random device variations guarantee there are periods where the data on the bus is invalid. The design tools cannot change the physics. Designs must be tolerant of the fact that there are always periods when the data on any bus is invalid.

### 8.3.1 Synchronous Design.

To overcome the problem of not knowing when the data is valid, almost all hardware designs use a clock to define when the data is valid. This is known as synchronous design. In this design style, the data is valid for some time before the clock edge (setup time) and some time after the clock edge (hold time). The clock signal is generated from a master source and then distributed throughout the device. Special care must be taken so that the clock arrives to all functional blocks in the device at the same time. Delivery of the clock to different devices at different times is known as clock skew. FPGAs contain a limited number of specialized trees that can be used to minimize clock skew. Although the design tool attempts to recognize clock trees, the designer must often explicitly declare these trees so that the synthesis tool will correctly accommodate them. The clock trees are often heavily loaded, driving many devices while the data lines often only drive a single device. This means the data is often naturally too fast and that the synthesis tool must incorporate delays to allow the device to meet timing. These delays are often created by inserting additional buffers in the data signal path or by artificially loading the data. This delay hardware is inserted without notifying the designer. Speed differences between the clock and the data path result in the failures listed in figure 10 as fast path (the data arrives too soon) and slow path (the data is too late). These failures are especially sensitive to variations in temperature and voltage.

### 8.3.2 Synchronous Design—Multiple Clock Domains.

Ideally, a design will have only a single master clock. Unfortunately, modern designs commonly require several independent clocks used within a single system. When signals move from one clock domain to another, special circuits and analyses are required. The NASA FPGA design guidelines point out that the synchronizers that are required to allow signals to cross clock domain boundaries have a nonnegligible failure rate, which must be considered when calculating the device reliability. Correct operation of circuits crossing clock domain boundaries cannot be guaranteed by simulation because the timing between different clock domains can vary arbitrarily. This would require an infinite number of simulations to verify. Special design techniques are used to allow signals to cross between clock domains, and designers will insert

them anytime they need them. The problem is that sometimes signals cross clock domains in obscure ways that are missed by the designers. Design tool vendors are attempting to address this problem by automatically identifying where signals cross clock domains and flag them. More advanced tools offer more comprehensive verification by performing code analysis for signals crossing clock domains, verifying correctness of synchronization between domains, and determining circuit behavior by injecting effects of metastability into the simulation [64].

### 8.3.3 Asynchronous Designs.

The most risky of all design styles is asynchronous design, where inputs and/or outputs are allowed to vary without respect to any clock. Asynchronous circuits are subject to a condition called “metastability,” in which signals transition from one value to another via quasi-stable states exhibiting an intermittent failure. Neither simulation (testing logic function) nor static timing analysis (testing single clock domain) can detect such a failure.

A typical example of such a situation is when the clock and data inputs of a flip-flop change values at approximately the same time. This leads to the flip-flop output oscillating and not settling to a value within the appropriate delay window. It happens when there is communication between discrete systems using different clocks. Experienced designers mitigate the event by adding synchronization between clock domains and isolating the “metastable” output to reduce propagation effects.

This state introduces a delay that varies, depending on the exact timing of the inputs. This delay can only be analyzed statistically. It is not possible to prevent an error from occurring; the best that can be done is to limit its probability. Although most designers avoid asynchronous design, there are cases where asynchronous inputs must be accepted. An example where an asynchronous design is required would be a reset path that is required to operate, even if the synchronizing clock is not present.

## 8.4 WIDE DATA BUSES AND DATA PATTERN-DEPENDENT ERRORS.

Modern, safety-critical systems can contain data and address buses that are 32, or even 64, bits wide. A wide variety of signaling standards are used in AEH. In single-ended signaling, the I/Os share a common power and ground connection. If all, or many, of the I/Os connected to the common power supply or ground change state simultaneously, a large spike in current will occur. This rapid change in current causes any parasitic inductances in the power supply and ground distribution network to have voltages induced across them that are proportional to the rate of change of the current. These parasitic inductances can be on the chip, in the device’s package, in the connection to the PCB, or in the PCB. These voltages, caused by the changing current, are known as supply/ground bounce and can be large enough to lead to erroneous circuit operation. The supply/ground bounce produced by simultaneously switching the outputs will be referred to as simultaneous switching noise.

Noise can also be introduced into the system via crosstalk between signals. Crosstalk coupling is primarily a function of the total inductance of the current path. This inductance is a function of the distance between the ground and power supply pins to the signal pin. Signal pins farther

away from a ground or power supply pin are more susceptible to noise. This problem is exacerbated when a number of I/Os in the region switch simultaneously.

Consider a block of logic that contains a large number of gates that simultaneously switch. If this logic is placed in close physical proximity during the design tool's place-and-route step, then it is possible that the internal FPGA power supplies will be unable to supply the necessary power and a localized voltage drop (IR drop) in power supply voltage can occur. This change in power supply voltage will result in changes in circuit timing and in the signaling levels used for the circuit. These changes can produce errors where the occurrence is data pattern-dependent.

Unless the underlying physics are understood, the conditions necessary to generate errors due to supply/ground bounce are difficult to identify. The parasitic elements needed to accurately model supply/ground bounce are rarely known and almost never used in logical simulations. Special targeted analysis is required to identify if supply/ground bounce is a concern in a design.

### 8.5 COMBINATIONAL FEEDBACK/QUASI-DIGITAL CIRCUITS.

FPGAs provide the user with the ability to configure the device in nearly an infinite number of ways. This flexibility can allow the designer to implement unexpected configurations.

For example, it is possible to configure an odd number of inverting gates into a circuit known as a ring oscillator. Inverters 1, 2, and 3 form the oscillator, while inverter 4 converts the analog sine wave back to a square wave (figure 12). This configuration has an output, but no inputs, and the timing is determined by the speed of the inverters and is not synchronized to any clock. This makes the ring oscillator very sensitive to temperature variations and this configuration is often used as a temperature sensor. When the hardware is operating as a ring oscillator, the signals do not switch between normal digital signal levels. The oscillator is essentially an analog device using the gain present in the logic gates to produce an oscillator. Most HDL simulators assume digital logic and are unable to correctly simulate this simple configuration. Many design tools prevent the user from implementing a combinational feedback configuration such as a ring oscillator. To guarantee the correctness of the tools, the designer's ability to produce such problematic configurations must be restricted.

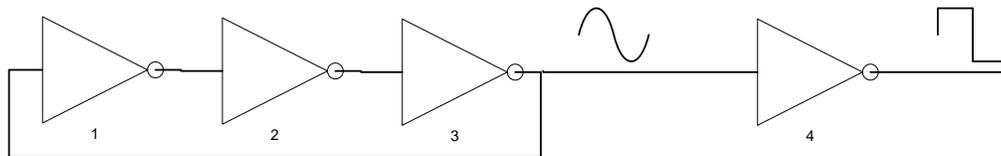


Figure 12. Ring Oscillator

### 8.6 SYNTHESIS ISSUES—WHAT THE TOOL REALLY BUILT.

The synthesis process is highly customizable and varies greatly from vendor to vendor. The variety of options and configurations makes it difficult for the designer to know exactly what the default synthesis settings are. Certain functions of synthesis, such as VHDL interpretation, are

standardized by the Institute of Electrical and Electronics Engineers [68]. However, nonstandard optimization techniques constitute the trade secrets of a given vendor. In fact, the tool user or designer does not know the details of synthesis algorithms and, therefore, is not aware of how the tool works. The magnitude of a change of the intended design in the synthesis process, and, thus, the impact on the final design, is not known. This impact depends upon the intricacies of the actual logic design, the selected tool used for synthesis, and the tool's current settings. Synthesis is not a standardized process. Each tool produces a unique implementation of the design. Due to concerns about IP and competitive advantage, it is not easy to publicize what synthesis algorithms are or what specific methods and techniques are used for simplification and optimization.

Creation of a placed-and-routed circuit from the HDL code that meets the performance goals is accomplished by merging logical and physical synthesis technologies. When such created designs cannot meet their realistic timing objectives, the solution is to use more traditional design methodologies. The intricacies of logical and physical synthesis are closely guarded IP of specific tool vendors. The general underlying background is well known, but the specifics of the synthesis algorithm are not.

### 8.6.1 Getting Less Than Expected.

The default configuration for almost all FPGA design tools is that all the compiler and synthesis optimizations are enabled. This can lead to unexpected implementations. For instance, a designer may write HDL code to specify a triple-redundant module (TRM), as shown in figure 13(a). However, the synthesis tool may determine that most of the hardware is redundant and implement the system, as shown in figure 13(b). The independent multipliers were identified as redundant and optimized away during synthesis.

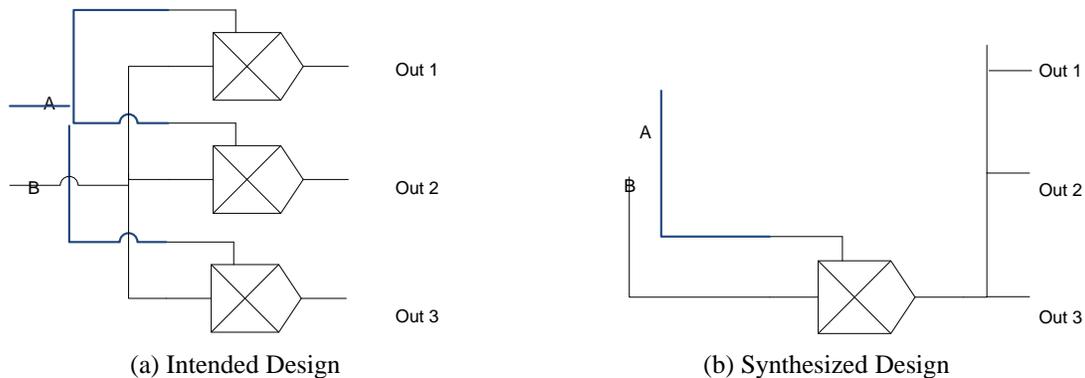


Figure 13. A TRM With Three Multipliers

### 8.6.2 Getting More Than Expected.

To meet timing, the synthesis tool will sometimes create redundant hardware to improve timing in what is called flip-flop replication. This can produce problems, especially in systems where some part of the circuit is attempting to monitor the performance of another circuit. In designs that are intended to be tolerant of SEU, it is common to have an output and monitor that are

guaranteed to be logical opposites of each other under all conditions. Consider the circuit of figure 14(a). In this circuit, the Output and the Monitor are always logically opposite. The logically equivalent implementation of the circuit could be generated by the synthesizer to help meet timing constraints and is presented in figure 14(b). In such a solution, an SEU of the top flip-flop will not affect the monitor output. Therefore, the resulting synthesized circuit does not guarantee that Output and Monitor are logical opposites, which defeats the purpose of the monitor output.

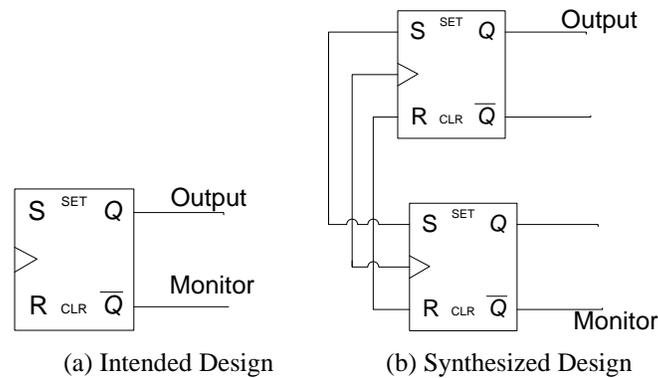


Figure 14. Flip-Flop Replication

## 8.7 HARDWARE THAT IS NONFUNCTIONAL IN NORMAL OPERATION.

The TRM (see section 8.6.1) and metastability (see section 8.3.3) occur when design optimizations are applied during synthesis and would not generate explicit warnings that the optimization had occurred. The biggest problem with this circuitry is in circuits that should not operate if the hardware is functioning normally. In this case, error may not be detectable on working hardware. The only method for guaranteeing correct operation of this type of circuitry is via simulation and assuring that the netlist includes the correct circuitry.

### 8.7.1 Synthesizer Optimizations.

Since the above problems are caused by the synthesizer performing optimizations, the designer could turn off all synthesizer optimizations. This is a possible solution, but it will be difficult to meet timing and area constraints without optimizations. Alternatively, an experienced designer would recognize these conditions ahead of time and configure the synthesizer optimizations appropriately. This is a possible solution, but difficult to verify if the designer has handled all possible areas of concern. The need for experienced personnel points to the need for a testing or design maturity model where the experience of the design and verification team is considered in determining the level of verification that must be demonstrated.

### 8.7.2 Gate-Level Verification.

The problems introduced by the synthesizer could be detected by running simulations using the gate-level netlist with back annotated timing data. In theory, running the full test suite on a gate-

level netlist is possible, but is very slow. However, DO-254 guidance states the design should not be analyzed at a level lower than the implementation was performed [1].

“Analyzing the implementation below the level of that specified by the designer, such as at the gate or transistor level, is not necessary ....”

Since the designer worked at the HDL level, not the gate level, DO-254 states that this level of verification is unnecessary.

### 8.7.3 Adding Test Circuitry.

It is desirable to catch errors in circuitry that does not operate during normal operation. To achieve this goal, the designer can add additional test circuitry to allow verification of normally nonfunctional circuits. However, it is difficult to anticipate the need for this extra test circuitry early enough in the design to include it in the requirements. These additional requirements will require addition verification tests, which could also need additional circuitry for testing.

## 8.8 RADIATION EFFECTS AND FPGA ARCHITECTURES.

Cosmic radiation enters the Earth’s atmosphere and collides with the atoms of the atmospheric gases. These collisions produce a wide variety of subatomic particles and most of these particles quickly recombine. However, significant quantities of high-energy neutrons are also produced by these collisions. Since neutrons possess no electrical charge, they do not recombine. The neutron flux is absorbed by the atmosphere. The greatest quantities of neutrons (called the neutron flux density) occur at an altitude of 60,000 feet; the quantity decreases as the altitude decreases. These high-energy neutrons can cause flip-flops and memory cells in modern semiconductor electronics to change state. This is shown in figure 15 and is known as an SEU.

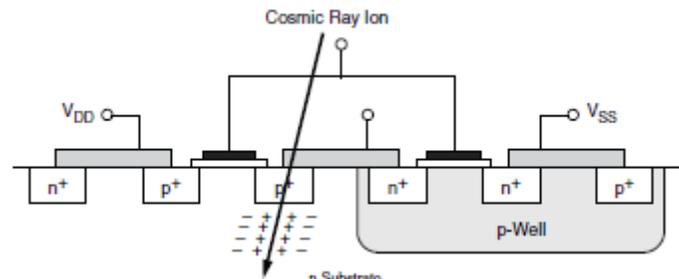


Figure 15. An SEU in an FPGA Using a CMOS Process [77]

Table 2 shows the expected SEU mean time to error for a large FPGA on a satellite in a geosynchronous orbit. Device configuration files are stored in the configuration memory. Errors in the configuration memory can potentially reprogram the device to produce some erroneous function. However, in a programmed FPGA, only a small fraction of the total configuration memory or block memory is used. Therefore, many errors may occur in these bits with no impact to circuit operation. A single event functional interrupt (SEFI) event is a detectable

device error. The probability that a random configuration or block memory error will result in an SEFI depends on the complexity of the design and the architecture used.

Table 2. The SEU Mean Time to Error for a Large FPGA in Geosynchronous Orbit [77]

| XRR2V6000 – 36,000 km | Mean Time to Error | Units |
|-----------------------|--------------------|-------|
| Configuration Memory  | 1.8                | Hours |
| Block Memory          | 11.8               | Hours |
| POR-SEFI              | 221                | Years |
| SMAP-SEFI             | 181                | Years |

Whereas Xilinx and Altera use SRAM-based configuration memory that is susceptible to neutron radiation, Actel uses a flash memory-based configuration memory that is far less susceptible to neutron radiation. Flash-based memory cells are substantially larger and more complex than SRAM-based memory cells. Therefore, there is a tradeoff in device capability and speed when using flash-based configuration memory. It should be noted that by using SEU mitigation techniques, such as TRMs, it is possible to harden an SRAM-based FPGA against radiation. SRAM-based configuration memory FPGAs have been used on numerous space missions with no radiation-induced problems.

#### 8.9 RADIATION—DO-254 AND DO-160.

Section 1.2 of DO-254 states that environmental criteria are beyond the scope of DO-254, but radiation-effect mitigation is often addressed using the system safety assessment process. Unlike other environmental conditions, such as humidity and electromagnetic noise susceptibility, there are no physical tests performed to guarantee correct operation of the device when subjected to radiation. Susceptibility to radiation effects is truly an environmental issue and would be best validated by actually irradiating the hardware under test to guarantee that the SEU mitigation scheme works correctly and can tolerate the expected radiation levels. From this point of view, radiation effects would best be handled as an environmental test that should be added to DO-160 testing.

#### 8.10 WHAT CIRCUIT IS BEING GENERATED.

A significant concern common to both the ASIC and FPGA flow is that details of the actual hardware implementation are protected as IP and are not available for inspection. In an FPGA, the programmable fabric consists of many programmable logic blocks (PLB). The overall function of the PLB and the external connection requirements are available for inspection, but the details of the hardware implementation of the PLB are unavailable. In the ASIC flow, a standard cell library is used to implement the RTL in hardware. Every cell in the library is characterized by how its performance varies with temperature, voltage, and loading; this data is used for the simulations. Standard cell libraries are usually sold or licensed for use as encrypted IP; therefore, the end-user cannot know the details of how the standard cells are implemented without description.

## 8.11 UNUSED INPUTS AND OUTPUTS.

FPGAs may contain a large number of pins that may be unused in any given application. These pins can include those that are used for manufacturing the FPGA, which need to be correctly configured for safe operation, as well as numerous other pins that are undefined pins in the design. Since the compiler and synthesizer treat unused pins as “don’t care,” these pins may be unpredictable in actual operation. Other issues are related to properly terminated inputs. FPGAs handle unused pins via software, exploiting the programmable nature of the microcircuit. However, the specific implementation details may differ. For example, in Actel SX and SX-S, special-purpose clock inputs do not have an output stage and must therefore be terminated by the user to prevent large currents. As another example, unused low-voltage differential signaling receiver inputs should be left unconnected, as advised by the documentation. Depending on the device, pins labeled “N/C” may be used for internal purposes and terminating them on the board may result in problems; conversely, not terminating N/C-labeled pins in certain cases can have adverse effects on system behavior. Some configuration pins have very high-value internal pull-up resistors and can be switched by high-speed signals at the board level. It is, therefore, critical to ensure that all pins are properly terminated to avoid parametric and long-term reliability effects. Some will affect the functionality of the chip, which may or may not be caught in testing [69].

## 8.12 OTHER CONSIDERATIONS.

For critical circuits, the designer must examine the output reports from the synthesizer very carefully. Common issues to check include states that cannot be exited, outputs of Gray code machines that can glitch, unintended flip-flop replication, and the desired/specified style not being implemented (sometimes the synthesizers automatically substitute one type of state machine for another). The actual state machine implemented for the same HDL code can vary dramatically with different design tools. The designer should always examine the generated design carefully. For instance, it has been observed that sometimes the logic will explode with excessive gates. Reset path timing usually has a problem involving race conditions. If some circuits are reset too soon, it can prevent the reset signals from propagating to other places they need to go. The required timing for the reset logic depends on the internal logic delays and is not synchronized to the clock. Differences in the temperature, power supply voltage, or signal levels can change the timing of the reset paths. To eliminate these timing variations, logic operating on the opposite clock edge is often inserted to prevent these race conditions. While solving the variation problem, the designer now must complete all operations in one-half of a clock cycle, rather than a full clock cycle. This leads to the tight timing the designer needs to consider at the gate level. Note that VHDL does not cover physical states, just logical ones. The HDL does not know if it is a one-hot-, or binary-, or gray-coded implementation and what flip-flops have been replicated during synthesis. Such issues are not detectable at the black box simulation level nor by checking Boolean equations for logical equivalence.

### 8.13 POWER UP/RESET ISSUES.

When an FPGA or ASIC is either powered up or comes out of reset, there is often a period of time when the device outputs are unpredictable. The performance of a component during power up is difficult to predict, as there are often multiple power supplies to the part that will turn on in an uncontrolled fashion. If the output drivers receive power before the internal logic, all of the glitches produced by the internal logic can be sent through the outputs to other devices in the system. Even a normal reset can contain internal race conditions that can produce periods where the outputs are unstable. The Wide-Field Infrared Explorer spacecraft was lost when the outputs of an FPGA produced unexpected outputs during power up. The unexpected output resulted in the system reset process not completing, which led to the early firing of a pyrotechnic device and ultimately to the failure of the mission [70].

### 8.14 WHAT CAN BE DONE TO PREVENT PROBLEMS.

The human factor increases the chance of error or misinterpretation. HDL used for hardware development can be classified as a computer programming language, with some unusual constructs to account for the parallel nature of the hardware implementations. The developer applies the knowledge of the language syntax and structure to describe his or her design idea. The challenge is to force the conceptual vision of the intended circuit into rigorous constraints of the HDL. A deep knowledge of the hardware, knowing what is actually produced as the result of specific HDL structures, is critical for successful implementation of designs.

Manufacturers take many steps to assure that the designs produced are correct on the first design iteration. The first steps are to assure that the RTL code, as written, is fully specified. The idea is to not leave anything to the discretion of the compiler. The output of the RTL code for “don’t care” and illegal conditions should be completely specified. In addition, a project-wide (possibly corporate-wide) programming style is defined to prevent the usage of risky or problematic RTL programming techniques. These rules are enforced by using linting programs that check the RTL style for rule violations. Linting programs act much like an experienced programmer looking over one’s shoulder.

### 8.15 DESIGN ISSUES SUMMARY.

Logic designers often replicate logic for reliability or performance reasons. For example, if the load on an output is too high, then the load will often be split between multiple drivers (in some cases, outputs may be joined together, but this is not preferred and is usually avoidable). In other cases, cutting the load and duplicating the driver can help improve timing by distributing the capacitive load. The replication of combinational logic is quite straightforward. However, if this concept is extended to sequential logic and FSM design, then the situation is trickier, since state information is involved. Indeed, the logic may present different information to different parts of the circuit and, for example, may be inconsistent in the presence of a transient fault such as an SEU. The logical flip-flop can present different values to different parts of the circuit, depending on which physical flip-flop they are connected to. This is a call for caution in high-reliability

applications. Software CAD tools are more than happy to generate circuits of this class, while not generating logic to ensure self-consistency [71].

Despite valiant efforts to assure error-free design, it is still possible for the tool to produce an incorrect design. This can be caused by a variety of factors, including power distribution problems, signal coupling, and interference problems. It should be noted that some design tool vendors have tool suites that can be used to address these issues. These tools are usually poorly integrated with the rest of the tool flow and many users are unaware of the tools' existence.

One critical aspect for consideration is that the HDL is not the design; it is simply the designer's description of the desired logic. Running HDL simulations and test benches is insufficient proof of a design's correctness. The design is then converted from an HDL description to a logical description, which is then mapped to the hardware on the FPGA. Then the design is physically located and wired in the device during the place-and-route step. Until the place-and-route step has occurred, the actual timing of the device is only estimated. The fidelity of the actual design to the intended design depends on the quality of the synthesizer, which is enigmatic, and the ability of the designer to:

- Write synthesizable HDL.
- Understand the synthesis process and tool employed.
- Control the synthesis process.
- Verify that the synthesis process produced what was intended.
- Correctly guide the back-end, place-and-route tools.

The danger in the process of converting HDL into a hardware implementation is that there are many logically equivalent ways to represent any single design. The tool may implement the design using a representation contrary to the designer's intent. Problems can arise from the synthesizer replicating, or combining circuits in an undesirable manner, or eliminating logic that the synthesizer believes is redundant. The actual implementation of a design into hardware requires translating an abstract description of the design into actual hardware. While not as abstract and complex as logic synthesizers, failure to understand the processes in the place-and-route process has the potential to cause design errors.

A significant danger to AEH design assurance is using outdated traditional techniques of directed test and code coverage. New effective techniques, widely adopted in other industries, have been developed to facilitate AEH verification. These include constrained random test generation, assertion-based verification, formal model checking, clock domain crossing analysis, unified coverage, verification management, and requirement tracking. These better verification techniques address many of the issues addressed above and help developers to achieve early defect removal and simplification of testing activities [73]. It should be noted that all the above techniques reduce the probability of an error in the hardware, but none are sufficient to guarantee that the physical implementation of the hardware is correct.

## 9. FINDINGS AND RECOMMENDATIONS.

The survey of tool users conducted for this research found that 17% of the tool users had experienced tool errors that they considered significant and numerous. However, when asked to list their satisfaction with the tools on a scale from 1 to 5, with 5 being completely satisfied, over 96% of the respondents marked their level of satisfaction with the tools as a 4 or higher. Further investigation into this apparent contradiction indicated that many of the reported tool errors would not impact the safety of the final design. Some examples of non-safety-related tool errors included tool features not operating as the salesperson represented and tool software crashes. During the entire research period, the authors continually canvassed tool users about any tool errors that they encountered. Occasionally, what the tool users identified as a tool error were actually errors resulting from incorrect application of the tool. The authors were unable to identify a single case of a tool producing an error that resulted in an error in the final design. The authors believe the high level of satisfaction with the tools reflects a satisfaction with the tool outputs.

Preventing errors from reaching a complex final product is a difficult goal to achieve. Even tools that are known to produce correct results can introduce errors if the tools are applied incorrectly. There are always humans in the design and verification process, and these humans may not fully understand the limitations of the tools that they are using. For example, the test cases demonstrated failures that occurred due to problems related to the inductance of the power supply network. Tools that analyze the effect of inductance and resistance in the power supply networks of complex devices require a tool suite far different than the tools used in a normal HDL design process. Even if the user correctly anticipated the need for the new tools, these tools may require expertise beyond what an HDL designer would possess.

The following list identifies the recommendations of this research:

- The outputs of the primary design or verification tools used for level A, B, and C safety-critical designs should always be independently assessed.
- Independent assessment of the design tool's outputs should not be viewed as a single event, but instead as a series of overlapping, independent assessments.
- Independent assessment events require a process where all tests and their results are documented and fully analyzed. Independent assessment events can include:
  - Verification testing
  - Simulations
  - Debugging
  - Hardware verification
  - System validation

- Which independent assessments are applicable to a specific design will vary with the design flow, so it is difficult to require that any particular independent assessment be performed.
  - The amount of independent assessment required can be varied, depending on the DAL. For instance, a level C might require a single independent assessment of the working hardware, while a level A might require independent assessments while the design is at the HDL level, when the design is a gate-level netlist, and then on the actual hardware.
- If there is no independent assessment of a tool's outputs, DO-254 currently allows a relevant tool service history to be used to avoid tool qualification. The authors believe that tool service history is a poor indicator of a tool's ability to produce a correct design. Because the authors believe that independent assessment should always occur for level A, B, and C designs, service history should not be used to avoid tool qualification.
- Many tools undergo minor improvements throughout their life. Designers should be allowed, and perhaps encouraged, to use these improved versions of the tools for their designs. It is probably unwise to be the first users of a tool that has undergone a major software revision, but it may make sense if the new tool has new capabilities that will enhance the safety of the final design.
- Tool qualification should be limited to the exceptionally rare case where independent assessment of the tool output is impractical or infeasible.
- In a complex design, it is impossible to generate a comprehensive verification suite. Constrained random verification should be used to increase the number of test cases generated and thus increase the number of errors detected by the test cases.
  - DAL A and B designs should also be required to use constrained random verification methods.
- Assertions should be used to increase the observability of the errors that are generated.
  - DAL A and B designs should be required to use assertions for all requirements and derived requirements that can be addressed by assertions.
- Linting tools should be required to enforce HDL coding standards.
- Level A and B designs require an experienced design team. A team could demonstrate expertise by executing a level C design before being allowed to attempt level A and B designs.

Even if the design and verification tools can prove that a design is functionally correct under all conditions that were considered, design errors in the hardware can still occur due to unforeseen conditions. The best way to avoid these errors is to have an experienced design and verification staff with the appropriate expertise. This will allow the team to identify potential problems while

still in the design phase and allow error mitigation techniques to be incorporated. In addition, an experienced staff writing the requirements will carefully detail the logical and timing functions of the system for both normal operation and error conditions.

Because obscure errors can slip past even the best design and verification teams using the highest quality tools, it is the opinion of the authors that the outputs of the primary design or verification tools used for level A, B, and C safety-critical designs must always be independently assessed.

## 10. REFERENCES.

1. DO-254, "Design Assurance Guidance for Airborne Electronic Hardware," RTCA Inc., Washington, DC, April 19, 2000.
2. CAST-27, "Clarifications on the Use of RTCA Document DO-254 and EUROCAE Document ED-80, Design Assurance Guidance for Airborne Electronic Hardware," June 2006.
3. CAST-28, "Frequently Asked Questions (FAQs) on the Use of RTCA Document DO-254 and EUROCAE Document ED-80, Design Assurance Guidance for Airborne Electronic Hardware," December 2006.
4. CAST-30, "Simple Electronic Hardware and RTCA Document DO-254 and EUROCAE Document ED-80, Design Assurance Guidance for Airborne Electronic Hardware," August 2007.
5. FAA AC 20-152, "RTCA, Inc., Document RTCA/DO-254, Design Assurance Guidance for Airborne Electronic Hardware," June 30, 2005.
6. DO-178B (EUROCAE ED-12B), "Software Considerations in Airborne Systems and Equipment Certification," RTCA Inc., Washington, DC, 2001.
7. Pampagnin, P. and Menis, J.F., "DO254-ED80 for High Performance and High Reliable Electronic Components," Internal paper, Barco-Siles S.A., Peynier, France, 2007.
8. Henson, J., "Equivalence Checking for FPGA Design," *Mentor Graphic Technical Publication*, May 2007.
9. Berens, K., "Programmable Logic Devices—Building the Case for Software-Style Assurance," Software Assurance Symposium (SAS), Morgantown, West Virginia, 2003, slide #3, [http://www.nasa.gov/centers/ivv/ppt/172554main\\_Berens\\_Assurance\\_Programmable\\_Logic\\_Devices\\_SAS2003.ppt](http://www.nasa.gov/centers/ivv/ppt/172554main_Berens_Assurance_Programmable_Logic_Devices_SAS2003.ppt).
10. SAE ARP 4754/EUROCAE ED-79, "Certification Considerations for Highly Integrated or Complex Aircraft Systems."

11. SAE ARP 4761, "Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems."
12. Dunar, A. and Waring, S., *The Power to Explore*, Chapter 12, June 1, 2009, <http://history.msfc.nasa.gov/book/chpttwelve.pdf> (last accessed 1/29/10).
13. Halfhill T., "An Error in a Lookup Table Created the Infamous Bug in Intel's Latest Processor," *BYTE Magazine*, March 1995.
14. Flight 501 Inquiry Board, "Ariane 5 Flight 501 Failure," Paris, July 1996, <http://sunnyday.mit.edu/accidents/Ariane5accidentreport.html> (last accessed 1/29/10).
15. McCormick, L., "A Short History of the Airbag," June 1, 2009, [http://www.consumeraffairs.com/news04/2006/airbags/airbags\\_invented.html](http://www.consumeraffairs.com/news04/2006/airbags/airbags_invented.html) (Last accessed January 29, 2010).
16. Staab, Thomas C., "Using SW-TMM to Improve the Testing Process," *Crosstalk: The Journal of Defense Software Engineering*, November 2002.
17. Burnstein, I., Homyen, A., Grom, R., and Carlson, C.R., "A Model to Assess Testing Process Maturity," *Crosstalk The Journal of Defense Software Engineering*, November 1998.
18. Rosebrugh, C., "Using Vera and Constrained Random Verification to Improve DesignWare Core Quality," June 1, 2009, <http://www.design-reuse.com/articles/9818/using-vera-and-constrained-random-verification-to-improve-designware-core-quality.html> (last accessed January 24, 2014).
19. Cooley, J., "The 2007 DeepChip Verification Census (Part 1 & 2) or A Census of 18 Engineers on Design Verification Tool Use," DeepChip.com Website, April 2007, <http://www.deepchip.com/posts/ducon07.html>, (last accessed January 24, 2014).
20. Aljer, A. and Devienne, P., "Co-Design and Refinement for Safety Critical Systems," *Proceedings of the 19th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, IEEE, 2004, pp. 78-86.
21. Bannow, N. and Haug, K., "Evaluation of an Object-Oriented Hardware Design Methodology for Automotive Applications," *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, Paris, February 2004, Vol. 3, pp. 268-273.
22. Bhatt, D. et al., "Model-Based Development and the Implications to Design Assurance and Certification," *Proceedings of the 24th Digital Avionics Systems Conference*, October 30-November 3, 2005.

23. Bunker, A., Gopalakrishnan, G., and McKee, S.A., "Formal Hardware Specification Languages for Protocol Compliance Verification," *ACM Transactions on Design Automation of Electronic Systems*, Vol. 9, No. 1, January 2004.
24. Bunker, A., McKee, S.A., and Gopalakrishnan, G., "An Overview of Formal Hardware Specification Languages," *Grace Hopper Celebration of Women in Computing*, 2002.
25. Camposano R. and Wilberg J., "Embedded System Design," *Design Automation for Embedded Systems*, Vol. 1, No. 1-2, January 1996, pp. 5-50.
26. Chee, W.L., Zain, Ali, N.B., and Nair, R.S., "Design of Low Cost FPGA Based PCI Bus Sniffer," *Proceedings of the IEEE International Conference on Field-Programmable Technology*, Tokyo, December 2003, pp. 420-423.
27. Cooper P.A., "Lessons Learned Using Software-Assisted Systems Engineering on Large Satellite Development Contracts," *IEEE Aerospace and Electronic Systems Magazine*, Vol. 21, No. 5, May 2006, pp. 7-11.
28. Dajani-Brown, S., Cofer, and D.' Bouali, A., "Formal Verification of an Avionics Sensor Voter Using SCADE," *Proceedings of the Joint International Conference on Formal Modelling and Analysis of Timed Systems, and Formal Techniques in Real-Time and Fault-Tolerant Systems*, Lecture Notes in Computer Science, Vol. 3253, pp. 5-20.
29. Hayek, A. and Robach, C., "From Specification Validation to Hardware Testing: A Unified Method," *Proceedings of the International Test Conference (ITC '96)*, October 1996, pp. 885-894.
30. Hilton, A.J., "High-Integrity Hardware-Software Codesign," Ph.D. Thesis, The Open University, April 2004.
31. Hilton, A. and Hall, J.G., "On Applying Software Development Best Practice to FPGAs in Safety-Critical Systems," *Proceedings of the 10th International Conference on Field-Programmable Logic and Applications*, Villach, Austria, August 2000.
32. Hilton, A.J. and Hall, J.G., "Developing Critical Systems With PLD Components," *Proceedings of the 10th International Workshop on Formal Methods for Industrial Critical Systems (FMICS)*, September 2005.
33. Hilton, A.J., Townson, G., and Hall, J.G., "FPGAs in Critical Hardware/Software Systems," *Proceedings of the ACM/SIGDA 11th International Symposium on Field Programmable Gate Arrays*, Monterey, California, ACM, 2003, p. 244.
34. Hoskote, Y.V. et al., "Automatic Verification of Implementations of Large Circuits Against HDL Specifications," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, March 1997, pp. 217-228.

35. Karlsson, K. and Forsberg, H., "Emerging Verification Methods for Complex Hardware in Avionics," *Proceedings of the 24th Digital Avionics Systems Conference*, October-November 2005, Vol. 1, pp. 6.B.1-61-12.
36. Kern, C. and Greenstreet, M.R., "Formal Verification in Hardware Design: A Survey," *ACM Transactions on Design Automation of Electronic Systems*, Vol. 4, No. 2, 1999, pp. 123-193.
37. Marcon, C.A.M. et al., "Prototyping of Embedded Digital Systems From SDL Language: A Case Study," *Proceedings Seventh IEEE International High-Level Design Validation and Test Workshop*, Cannes, France, October 2002, pp. 133-138.
38. Mencer, O., "ASC: A Stream Compiler for Computing With FPGAs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 25, No. 9, September 2006, pp. 1603-1617.
39. Mills, M. and Peterson, G., "Hardware/Software Co-Design: VHDL and Ada 95 Code Migration and Integrated Analysis," *Proceedings of the Annual ACM SIGAda International Conference on Ada*, Washington, DC, 1998, pp. 18-27.
40. Miner, P.S. et al., "A Case-Study Application of RTCA DO-254: Design Assurance Guidance for Airborne Electronic Hardware," *Proceedings of the 19th Digital Avionics Systems Conferences*, Vol. 1, pp. 1A1/1-1A1/8.
41. Nehme, C. and Lundqvist, K., "A Tool for Translating VHDL to Finite State Machines," *Proceedings of the 22nd Digital Avionics Systems Conference*, Vol. 1, pp. 3.B.6-1-7.
42. Peterson, G.D. and Hines, J.W., "Advanced Avionics System Development: Achieving Systems Superiority through Design Automation," *Proceedings of the 1998 IEEE Aerospace Conference*, 1998, Vol. 1, Issue 21, pp. 231-238.
43. Salzwedel, H., "Mission Level Design of Avionics," *Proceedings of the 23rd Digital Avionics Systems Conference*, Vol. 2, pp. 9.D.2-91-10.
44. Sangiovanni-Vincentelli, A., "Quo Vadis, SLD? Reasoning About the Trends and Challenges of System Level Design," *Proceedings of the IEEE*, Vol. 95, No. 3, March 2007, pp. 467-506.
45. Turner, K.J. and He, J., "Formally-Based Design Evaluation," *Proceedings of the 11th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, Lecture Notes in Computer Science, Vol. 2144, pp. 104-109.
46. Lange, M., *Assessing the ModelSim Tool for Use in DO-254 and ED-80 Projects*, Rev. 1.1, Mentor Graphics Corp., May 2007.

47. Berens, K., "NASA Complex Electronics Guidebook for Assurance Professionals," December 2004.
48. A380 Certification Review Item F-09, Issue 2, Digital Devices Design Assurance, March 2003.
49. Hilton, A. and Hill, J., "On Applying Software Development Best Practices to FPGAs in Safety-Critical Systems," Field Programmable Logic and Applications: The Roadmap to Reconfigurable Computing, 10th International Conference, FPL 2000 Villach, Austria, August 27-30, 2000 Proceedings, Lecture Notes in Computer Science, Volume 1896, pp. 793-796, Springer Berlin Heidelberg, 2000.
50. Young, D., "RTCA/DO-254: No Hiding Place for Avionics Suppliers?" *VMEbus Systems*, February 2004.
51. Hilderman, V., and Baghai, T., "Avionics Hardware Must Now Meet Same FAA Requirements as Airborne Software," *COTS Journal*, September 2003.
52. Thornton, R.K., "Review of Pending Guidance and Industry Findings on Commercial Off-the-Shelf (COTS) Electronics in Airborne Systems," FAA Report DOT/FAA/AR-01/41, August 2001.
53. Lee, C., "IPT Guidance for Acquisition of Systems With Complex Programmable Hardware Using DO-254," ERA Technology Ltd, June 2007.
54. Baghai, T. and Burgaud, L., "DO-254 Package: Process and Checklists Overview and Compliance With RTCA/DO-254 Document," March 2004.
55. Burgaud, L., "The DO-254 Users Group: A Proactive Initiative to Federate Industry Efforts," Presentation at the *FAA Software & AEH Conference*, New Orleans, Louisiana, July 2007.
56. Lundquist, P., "Certification of Actel Fusion According to RTCA DO-254," Master Thesis, Report LiTH-ISY-EX-ET-07/0332-SE, Linköping University, Sweden, May 4, 2007.
57. Jacklin, S. et al., "Development of Advanced Verification and Validation Procedures and Tools for the Certification of Learning Systems in Aerospace Applications," *Proceedings of the AIAA Infotech@Aerospace 2007 Conference and Exhibit*, Paper No. AIAA 2005-6912, Arlington, Virginia, September 26-29, 2005.
58. Crum, V., Homan, D., and Bortner, R., "Certification Challenges for Autonomous Flight Control Systems," *Proceedings of the AIAA Guidance, Navigation, and Control Conference and Exhibit*, Paper No. AIAA 2004-5257, Providence, Rhode Island, August 16-19, 2004.

59. Baghai, T. and Burgaud, L., *Reqtify: Product Compliance With RTCA/DO-254 Document*, May 2006.
60. Aldec, Inc., *DO-254 Hardware Verification: Prototyping With Vectors Mode*, June 26, 2007.
61. Leroy, J.E. and Bezamat, J., “Experience at Barco-Silex in FPGA Design With DAL C (DO254),” Internal Paper, 2007, Barco-Siles S.A., Peynier, France.
62. Pampagnin, P. and Menis, J., “DO254-ED80 for High Performance and High Reliable Electronic Components,” Internal Paper, Barco-Siles S.A., Peynier, France, 2007.
63. Dellacherie, S., Burgaud, L., and di Crescenzo, P., “Improve—HDL: A DO-254 Formal Property Checker Used for Design and Verification of Avionics Protocol Controllers,” *Proceedings of the 22nd Digital Avionics Systems Conference*, Indianapolis, Indiana, October 12-16, 2003, Vol. 1, pp. 1.A.1-1.1-8.
64. Lange, M., *Automating Clock-Domain Crossing Verification for DO-254 (and Other Safety-Critical) Design*, White Paper, Mentor Graphics Corporation, December 2007.
65. Cyliax, I., *The Foundation Environment*. *Circuit Cellar Online*, 2000, <http://www.circuitcellar.com/library/ropes/0100/c0100lrpdf.pdf> (last accessed 1/29/10).
66. Gardner, D., “Methods to Differentiate Mil/Aero Solutions Using FPGA,” *Proceedings of the 8th Annual MAPLD International Conference*, Washington, DC, September 7-9, 2007, Paper No. 145, Slide #30, <http://klabs.org/mapld05/abstracts/index.html> (last accessed 1/29/10).
67. Katz, R., “A Designer Engineer’s View,” Presentation at the *FAA 2005 SW&AEH Conference*, Norfolk, Virginia, July 2005.
68. *IEEE Std P1076-2002 Standard VHDL Language Reference Manual*, IEEE, 2002.
69. *Design Guidelines and Criteria for Space Flight Digital Electronics*, “Section I Special Pins,” NASA Office of Logic Design, 2004, [http://klabs.org/DEI/References/design\\_guidelines/nasa\\_guidelines/special\\_pins/special\\_pins.htm#unused\\_Inputs](http://klabs.org/DEI/References/design_guidelines/nasa_guidelines/special_pins/special_pins.htm#unused_Inputs), (last accessed 1/29/10).
70. Habinc, S., “Lessons Learned From FPGA Developments. Technical Report,” Gaisler Research, Goeteborg, Sweden, April 2002, [http://www.klabs.org/DEI/lessons\\_learned/esa\\_lessons/esa\\_fpga\\_001\\_01-0-0.pdf](http://www.klabs.org/DEI/lessons_learned/esa_lessons/esa_fpga_001_01-0-0.pdf) (last accessed 1/29/10).

71. "Design Guidelines and Criteria for Space Flight Digital Electronics," Section XII, Review of Digital Electronic Circuits, NASA Office of Logic Design, 2004, [http://klabs.org/DEI/References/design\\_guidelines/nasa\\_guidelines/review/review.htm](http://klabs.org/DEI/References/design_guidelines/nasa_guidelines/review/review.htm), (last accessed 1/29/10).
72. "Design Guidelines and Criteria for Space Flight Digital Electronics," Section IV. Finite State Machines, NASA Office of Logic Design, 2004, [http://klabs.org/DEI/References/design\\_guidelines/nasa\\_guidelines/fsm/finite\\_state\\_machines.htm#lockup\\_states\\_hdl](http://klabs.org/DEI/References/design_guidelines/nasa_guidelines/fsm/finite_state_machines.htm#lockup_states_hdl) (last accessed 1/29/10).
73. Mentor Graphics et al., "Understanding DO-254 and Solutions to Facilitate Compliance," June 1, 2009, [http://www.mentor.com/products/fv/techpubs/mentorpaper\\_35473.cfm](http://www.mentor.com/products/fv/techpubs/mentorpaper_35473.cfm).
74. Young, Duncan, "RTCA/DO-254: No Hiding Place for Avionics Suppliers?," VMEbus Systems, February 2004.
75. Kornecki, A. and Zalewski, J., "Experimental Evaluation of Software Development Tools for Safety-Critical Real-Time Systems," *Innovations in System Software Engineering*, Vol. 1, 2005, pp. 176-188.
76. Brady, D., Riggins, B., "Lessons Learned The Hard Way: FPGA-PCB Integration," Proceedings of the 6<sup>th</sup> Annual MAPLD International Conference, Washington, D.C., 2005, Paper No. 131, Slide #24, [klabs.org/mapld05/present/131\\_brady\\_bof-j.ppt](http://klabs.org/mapld05/present/131_brady_bof-j.ppt).
77. Xilinx Inc., "XAPP987, Single-Event Upset Mitigation Selection Guide," March 2008.

## 11. GLOSSARY OF TERMS.

**Component:** Self-contained part, combination of parts, subassembly, or unit that performs a distinct function of a system.

**Element:** An electronic device with terminals for connection to other electrical devices.

**Functional Failure Path:** The specific set of interdependent circuits that could cause a particular anomalous behavior in either the hardware that implements the function or the hardware dependent on the function.

**Gray Code:** A binary numerical system in which two successive values differ by only one digit.

**Linting:** A process of static analysis of source code flagging suspicious and nonportable constructs (derived from lint program in UNIX/C).

**Metastability:** A property of a digital circuit that results from input signals not being sufficiently stable immediately after a clock change and leads to an unpredictable future state.

Netlist: A software description of the connectivity of an electrical design.

Place-and-Route: A stage in the design of integrated circuits at which a layout of a larger block of the circuit or the whole circuit is created from layouts of similar subblocks; the process for a board is similar with varying levels of detail; the operation is usually performed by electronic design automation tools.

Testing Maturity Model: A maturity model with focus on testing inspired by the Software Engineering Institute Capability Maturity Model that assumes that there is a correlation between organizational maturity and the quality of software produced.

Validation: The process of determining that the requirements are the correct requirements and that they are complete.

Verification: The evaluation of an implementation of requirements to determine that they have been met.



## APPENDIX A—SURVEY QUESTIONNAIRE

# Programmable Logic Tools Questionnaire

### Introduction

This questionnaire is about your experiences and opinions concerning the use of programmable logic tools used to help design or verify airborne electronic hardware (AEH) like FPGA, PAL<sup>®</sup>, GAL, PLA, ASIC, or SoC on a DO-254 development program. *The questionnaire is intended for the individuals who have experience with developing or using such tools or experience qualifying such tools.* The purpose is to gather industry and certifying authority feedback on assessment and qualification of AEH programmable logic tools.

Your feedback will help establish critical issues and problems with AEH programmable logic tools and tool qualification and will help direct research in these areas. If you prefer, you may submit the survey anonymously (do not need to provide your or company names).

### Background Information

#### 1. What kind of organization do you work for?

- Avionics or engine control developer
- Aircraft or engine manufacturer
- Communications, navigation, or surveillance system developer for air traffic management
- AEH programmable logic tool developer
- Consultant
- Federal Aviation Administration
- Other government agency (please specify): \_\_\_\_\_
- Other, (please specify): \_\_\_\_\_

#### 2. What is your educational background? (check all that apply)

- |                                     |  |
|-------------------------------------|--|
| <input type="radio"/> technical     | <input type="radio"/> software                     |
| <input type="radio"/> non-technical | <input type="radio"/> control                      |
| <input type="radio"/> associate     | <input type="radio"/> electronics                  |
| <input type="radio"/> bachelor      | <input type="radio"/> mechanical                   |
| <input type="radio"/> master        | <input type="radio"/> computer science             |
| <input type="radio"/> doctoral      | <input type="radio"/> other (please specify) _____ |

#### 3. What is your role relevant to AEH Programmable Logic tools? (check all that apply)

- I am engaged in development of AEH programmable logic tools
- I am engaged in development of AEH programmable logic components
- I develop systems using AEH programmable logic components
- I verify systems using AEH programmable logic components
- I use AEH programmable logic tools
- I am a manager supervising DO-254 project
- I am an FAA engineer who approves tool data
- I am a Designated Engineering Representative (DER) who approves tool data
- Other, (please specify): \_\_\_\_\_

#### 4. What are your primary interests (check all that apply)

- |  |                                    |
|--|------------------------------------|
| <input type="radio"/> development          | <input type="radio"/> hardware     |
| <input type="radio"/> concept/architecture | <input type="radio"/> other: _____ |
| <input type="radio"/> verification         |                                    |

**4. How many years of experience?**

- <3
- 3-6
- 7-12
- >12

**AEH Programmable Logic Information**

The objective of this study is to provide the sponsor, the FAA, with input on potential safety issues in the assessment and qualification of tools used in developing airborne electronic hardware (AEH) for the aircraft. The devices in this category include components based on PAL (Programmable Array Logic), GAL (Generic Array Logic), PLA (Programmable Logic Array), EEPLD (Electrically-Erasable Programmable Logic Device), CPLD (Complex Programmable Logic Devices), FPGA (Field Programmable Gate Arrays), ASIC (Application Specific Integrated Circuits), System-on-a-Chip (SOC), and similar circuits used as programmable components of electronic hardware. This questionnaire is part of an ongoing research project funded by the FAA to help evaluate and clarify the tool assessment and qualification process defined in the Section 11.4 of the RTCA DO-254, “Design Assurance Guidance for Airborne Electronic Hardware”.

**1. What types of programmable logic devices are used (or considered for use) in your organization?**

*(check all that apply)*

- Field Programmable Gate Array (FPGA)
- Programmable Array Logic (PAL)
- Generic Array Logic (GAL)
- Programmable Logic Array (PLA)
- Erasable Programmable Logic Device (EPLD)
- Complex Programmable Logic Device (CPLD)
- System-on-a-Chip (SoC)
- Application Specific Integrated Circuit (ASIC)
- Other, (please specify): \_\_\_\_\_

**2. Which vendor’s programmable logic devices you use?**

|                                    | Name of the device/version  | Satisfaction level<br><i>(1 not satisfied to 5 extremely satisfied)</i> |
|------------------------------------|-----------------------------|---|
| <input type="radio"/> Actel®       | _____                       | _____   |
| <input type="radio"/> Atmel®       | _____                       | _____   |
| <input type="radio"/> Altera®      | _____                       | _____   |
| <input type="radio"/> Cypress®     | _____                       | _____   |
| <input type="radio"/> Lattice®     | _____                       | _____   |
| <input type="radio"/> Quick Logic® | _____                       | _____   |
| <input type="radio"/> Xilinx®      | _____                       | _____   |
| <input type="radio"/> other        | vendor: _____ product _____ | _____   |

**3. Which vendor’s programmable logic tool you use?**

|  | Name of the Tool/Version | Satisfaction level<br><i>(1 not satisfied to 5 extremely satisfied)</i> |
|--|--------------------------|---|
| <input type="radio"/> Synopsys®        | _____                    | _____   |
| <input type="radio"/> Intusoft™        | _____                    | _____   |
| <input type="radio"/> Mentor Graphics® | _____                    | _____   |
| <input type="radio"/> Tanner           | _____                    | _____   |
| <input type="radio"/> Cadence®         | _____                    | _____   |
| <input type="radio"/> Aldec™           | _____                    | _____   |
| <input type="radio"/> Novas            | _____                    | _____   |
| <input type="radio"/> Tau Simulation   | _____                    | _____   |
| <input type="radio"/> Synplify®        | _____                    | _____   |
| <input type="radio"/> Magma            | _____                    | _____   |
| <input type="radio"/> Verisity         | _____                    | _____   |
| <input type="radio"/> other            | vendor: _____ tool _____ | _____   |

**4. Please use the table below to list AEH tools you currently use or have used recently on a DO-254 program**  
*(if you have experience with more than three tools, please list the three most frequently used)*

| Tool Name and Version | Programmable Logic Tool Vendor | Type of Project (describe what it is used for; include typical project size in number of gates) | Phase of development used (check all)   | Has the tool been qualified?  |
|-----------------------|--------------------------------|---|---|---|
|                       |                                | <p>Satisfaction level: _____<br/> <i>(1 not satisfied, 5 extremely satisfied)</i></p>           | <input type="checkbox"/> Logic Synthesis<br><input type="checkbox"/> Physical Synthesis<br><input type="checkbox"/> Design<br><input type="checkbox"/> Simulation<br><input type="checkbox"/> Emulation<br><input type="checkbox"/> Timing Anal<br><input type="checkbox"/> Power Anal<br><input type="checkbox"/> Testing<br><input type="checkbox"/> Verification<br><input type="checkbox"/> Place/route<br><input type="checkbox"/> Integration<br><input type="checkbox"/> _____ | <input type="checkbox"/> yes, Level (circle one) A, B, C, D, not sure<br><input type="checkbox"/> no<br><input type="checkbox"/> don't know |
|                       |                                | <p>Satisfaction level: _____<br/> <i>(1 not satisfied, 5 extremely satisfied)</i></p>           | <input type="checkbox"/> Logic Synthesis<br><input type="checkbox"/> Physical Synthesis<br><input type="checkbox"/> Design<br><input type="checkbox"/> Simulation<br><input type="checkbox"/> Emulation<br><input type="checkbox"/> Timing Anal<br><input type="checkbox"/> Power Anal<br><input type="checkbox"/> Testing<br><input type="checkbox"/> Verification<br><input type="checkbox"/> Place/route<br><input type="checkbox"/> Integration<br>_____                          | <input type="checkbox"/> yes, Level (circle one) A, B, C, D, not sure<br><input type="checkbox"/> no<br><input type="checkbox"/> don't know |
|                       |                                | <p>Satisfaction level: _____<br/> <i>(1 not satisfied, 5 extremely satisfied)</i></p>           | <input type="checkbox"/> Logic Synthesis<br><input type="checkbox"/> Physical Synthesis<br><input type="checkbox"/> Design<br><input type="checkbox"/> Simulation<br><input type="checkbox"/> Emulation<br><input type="checkbox"/> Timing Anal<br><input type="checkbox"/> Power Anal<br><input type="checkbox"/> Testing<br><input type="checkbox"/> Verification<br><input type="checkbox"/> Place/route<br><input type="checkbox"/> Integration<br>_____                          | <input type="checkbox"/> yes, Level (circle one) A, B, C, D, not sure<br><input type="checkbox"/> no<br><input type="checkbox"/> don't know |

**5. Identify major criteria for evaluating new tools for selection to be used in DO-254 project? Rank the first five criteria assigning numbers from 1 (high) to 5 (low).**

- \_\_\_ Tool vendor reputation
- \_\_\_ Tool functionality
- \_\_\_ Acquisition cost
- \_\_\_ Compatibility with the existing tools in use
- \_\_\_ Compatibility with the development platform
- \_\_\_ Reliability/quality of the tool
- \_\_\_ Availability of vendor-supported training
- \_\_\_ Amount of training needed to use the tool
- \_\_\_ Amount/quality of documentation available
- \_\_\_ Quality of support and the access to the vendor's technical staff
- \_\_\_ Previous team familiarity with the tool
- \_\_\_ Tool performance on in-house internal evaluation
- \_\_\_ Host platform of the tool (workstation, operating system)
- \_\_\_ Compatibility with the selected programmable logic platform (FPGA, CPLA, ASIC, SOC)
- \_\_\_ Previous tool use on airborne product projects (under DO-254/DO-178B)
- \_\_\_ Tool performance (effort required, product quality in terms of size, power)
- \_\_\_ Ease of qualification
- \_\_\_ Other (please explain): \_\_\_\_\_

**6. When it comes to acquiring new programmable logic tools, which of the following apply prior to selection for a project?**

- We review the tool documentation, but do not test the actual tool relying on the vendor information
- We do limited tool familiarization (with e.g. demo version), but do not attempt extensive testing on a smaller project
- We extensively review and test the tool by training the personnel and using trial period on a smaller project
- We do formal independent third party assessment of the tool
- Other (please specify): \_\_\_\_\_

**7. Have you experienced successful/failed efforts to qualify programmable logic tools?**

- no
  - I don't know
  - yes, (please explain): \_\_\_\_\_
- \_\_\_\_\_
- \_\_\_\_\_
- \_\_\_\_\_
- \_\_\_\_\_

**8. If the tool assessment/qualification was performed/attempted, please comment**

- Clarity of the guidelines
 

|           |             |            |          |
|-----------|-------------|------------|----------|
| excellent | appropriate | sufficient | marginal |
|-----------|-------------|------------|----------|
- Ease of finding required information
 

|           |             |            |          |
|-----------|-------------|------------|----------|
| excellent | appropriate | sufficient | marginal |
|-----------|-------------|------------|----------|
- Increase of workload
 

|            |          |             |         |
|------------|----------|-------------|---------|
| negligible | moderate | significant | extreme |
|------------|----------|-------------|---------|
- Safety improvement
 

|             |            |          |          |
|-------------|------------|----------|----------|
| significant | noticeable | moderate | marginal |
|-------------|------------|----------|----------|

**9. Have you experienced finding errors (through the tool use or through the qualification process) in the programmable logic tools?**

- No errors, as far as I know, have been found
- Some errors have been found, but they have been few and minor
- Errors have been found that are significant or numerous
- I don't know

**10. What do you see as major issues regarding use of programmable logic tools?**

**11. What do you see as major issues regarding qualification of programmable logic tools?**

**12. What other programmable logic tools related experience or issues would you like to share?**

**OPTIONAL: Additional Information**

Name: \_\_\_\_\_

Company: \_\_\_\_\_

Phone: \_\_\_\_\_

E-mail: \_\_\_\_\_

*Note: If you'd rather submit your contact information in a different format (i.e., you don't want it attached to this survey), please send your contact information to Dr. Andrew J. Kornecki, ERAU as an e-mail with the subject "AEH TOOL HELP" [kornecka@erau.edu](mailto:kornecka@erau.edu)*

## APPENDIX B—SURVEY RESULTS

### B.1 GRAPHICAL SURVEY RESULTS.

This appendix includes additional graphical representations of the survey results, complementing section 3. Figures B-1 through B-5 address the survey population in terms of their experience, background, and interests. Figures B-6 through B-8 identify types of tools, devices, and tool vendors. Figures B-9 and B-10 present evaluation and selection criteria. Figures B-11 through B-16 show the qualification issues.

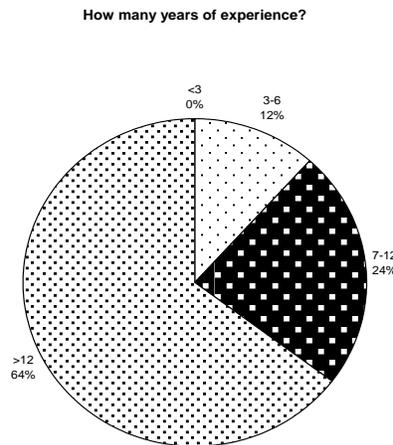


Figure B-1. Respondent Population—Experience

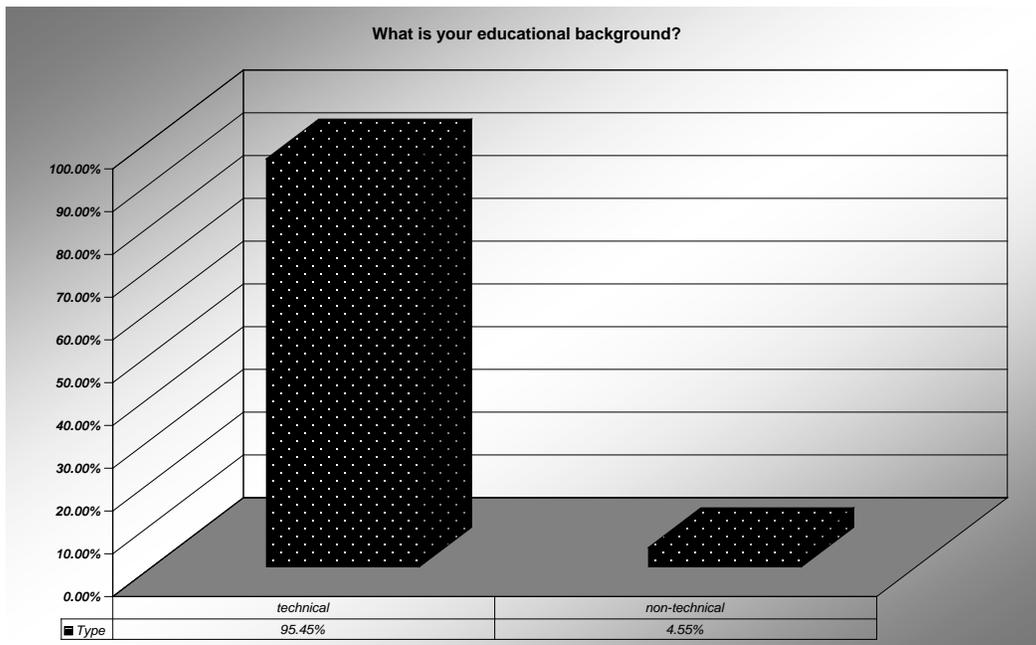


Figure B-2. Respondent Population—Educational Background: Type

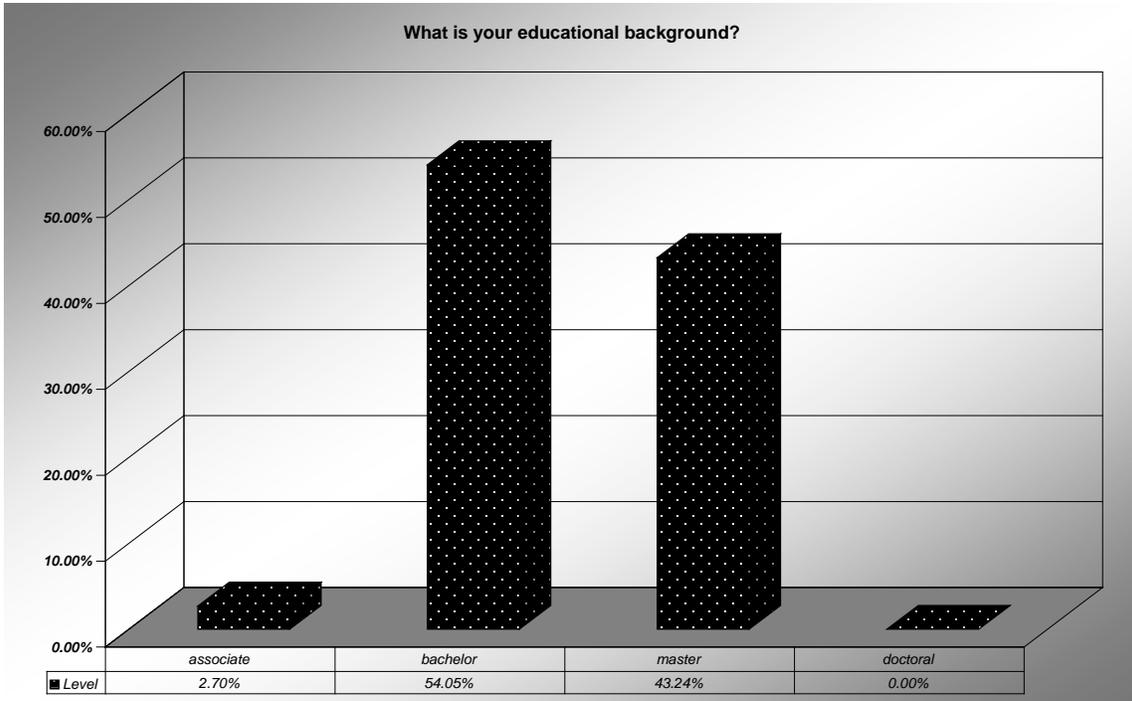


Figure B-3. Respondent Population—Educational Background: Level

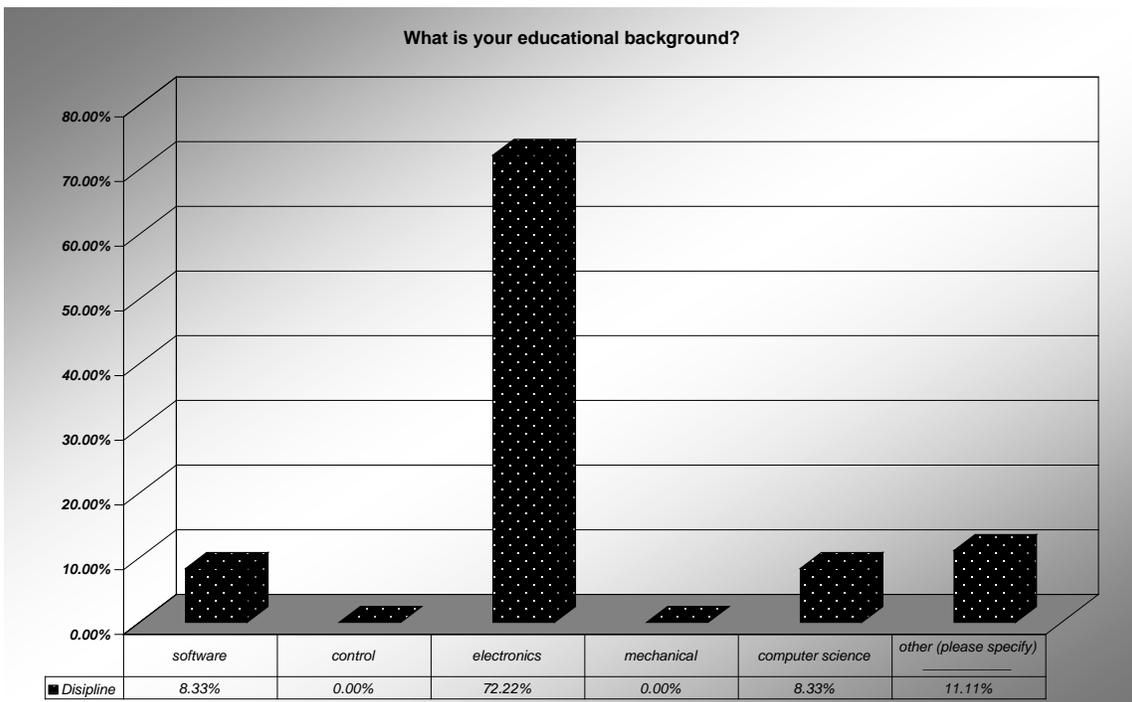


Figure B-4. Respondent Population—Educational Background: Major

What are your primary interests?

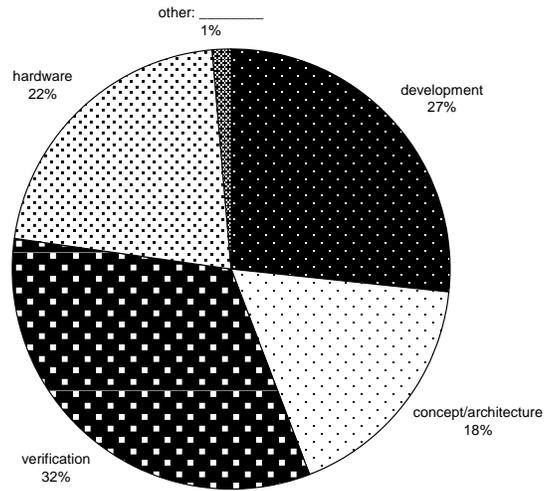


Figure B-5. Interest

What types of programmable logic devices are used (or considered for use) in your organization?

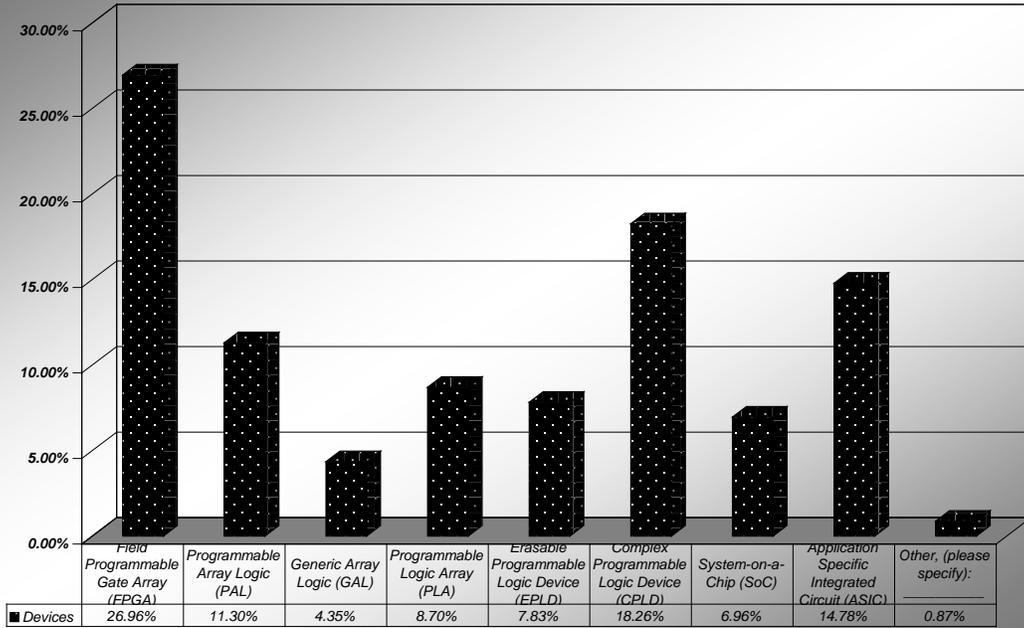


Figure B-6. Types of Programmable Logic Device

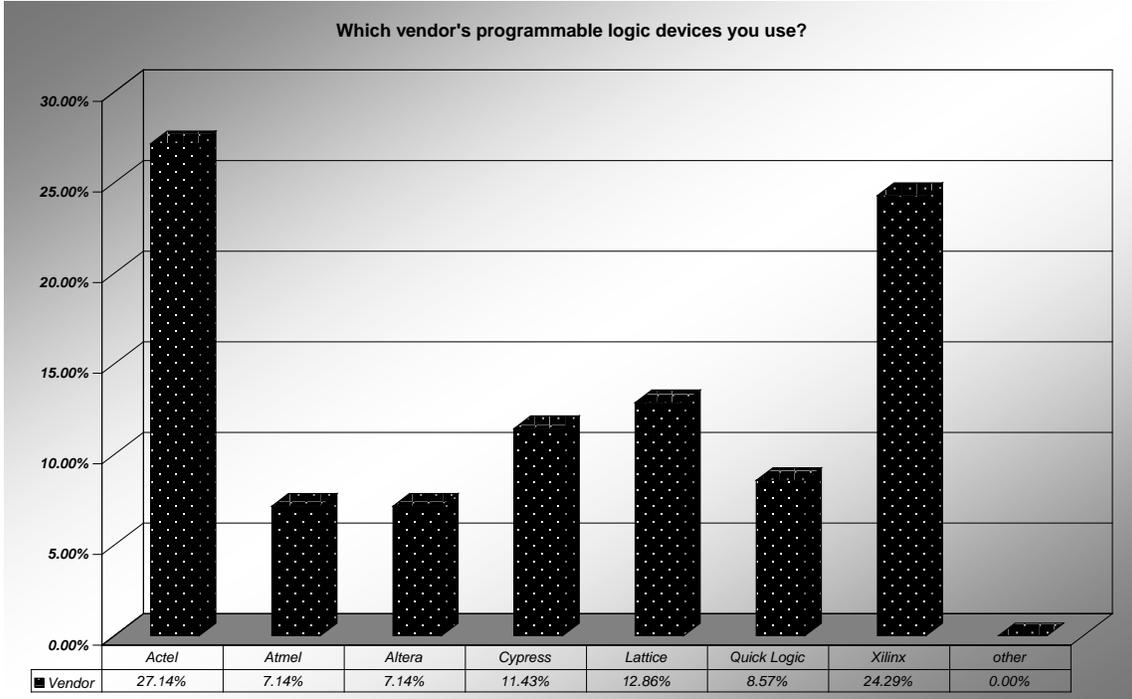


Figure B-7. Device/Hardware Vendors

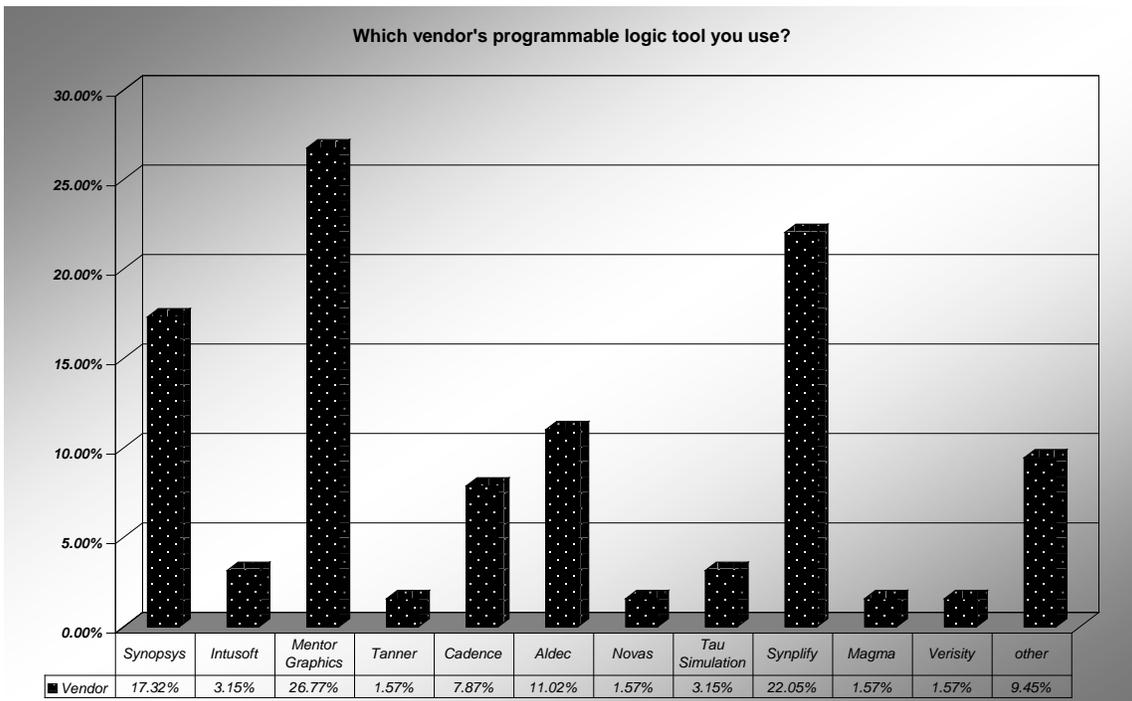
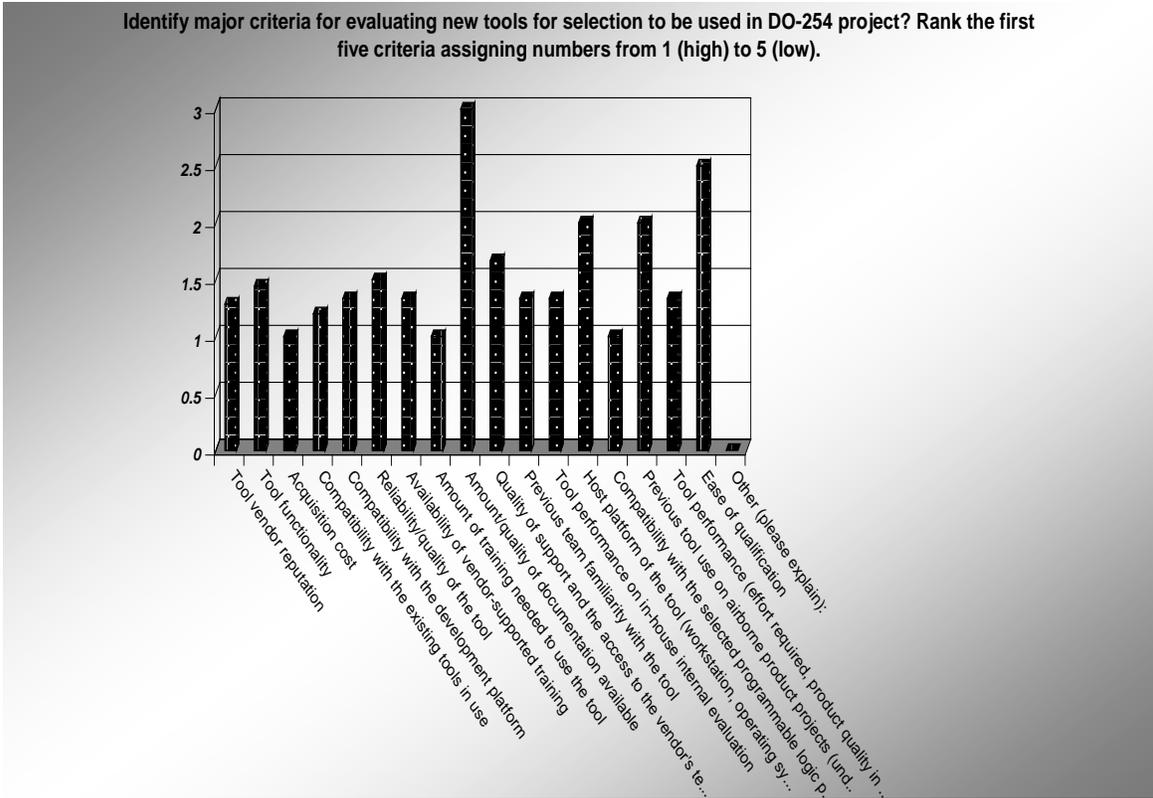


Figure B-8. Programmable Logic Device Tool Vendors



Note: Refer to item 5 from the questionnaire for a complete identification of the major criteria that are not fully shown in this figure.

Figure B-9. Evaluation Criteria

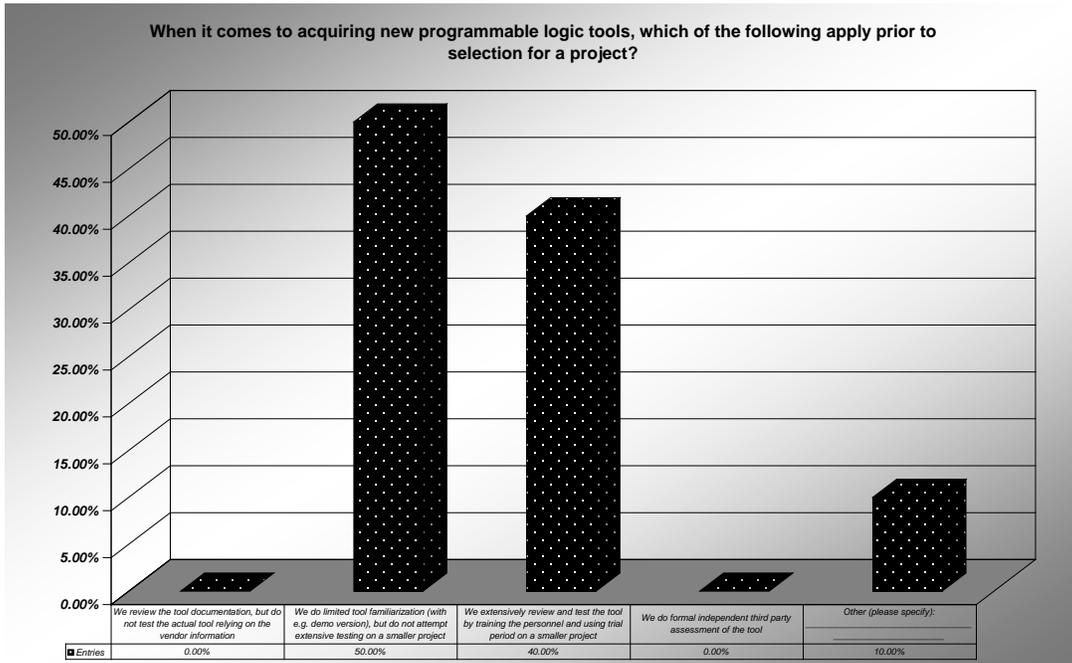


Figure B-10. Selection of Tools

Have you experienced successful/failed efforts to qualify programmable logic tools?

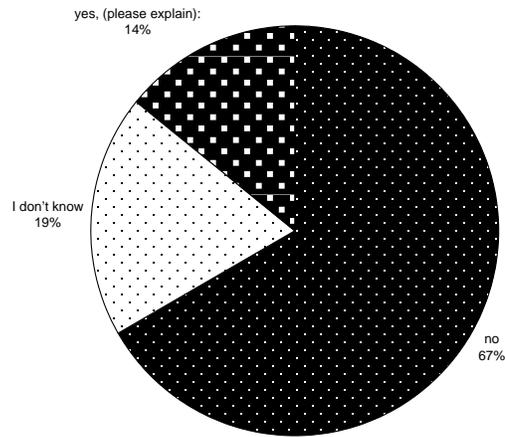


Figure B-11. Qualification Percentage

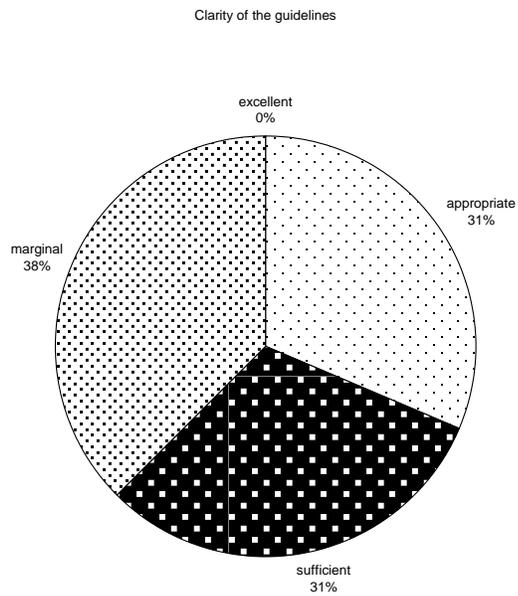


Figure B-12. Clarity of Guidelines

Ease of finding required information

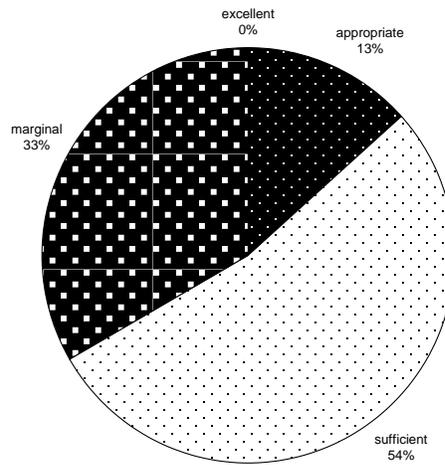


Figure B-13. Ease of Finding Information

Increase of workload

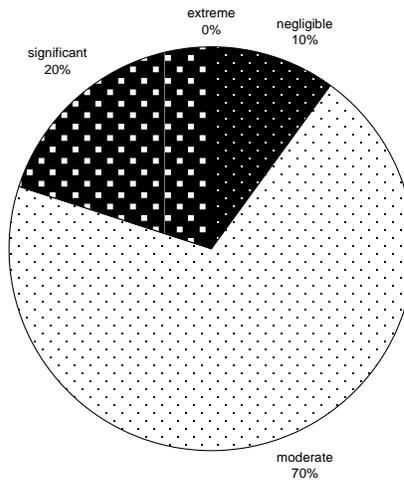


Figure B-14. Increase Workload

Safety improvement

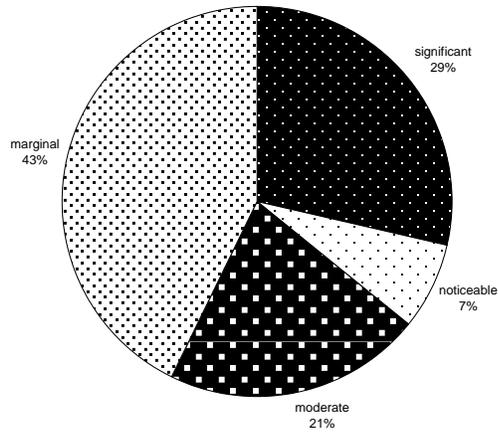


Figure B-15. Safety Improvement

Have you experienced finding errors (through the tool use or through the qualification process) in the programmable logic tools?

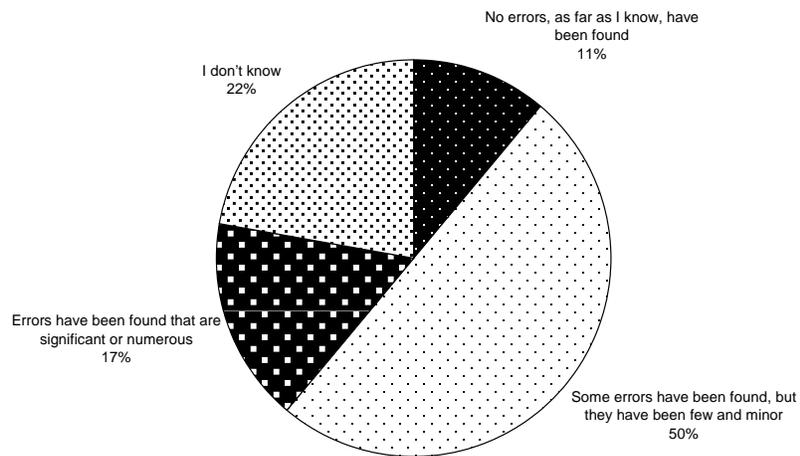


Figure B-16. Errors Found

## B.2 SURVEY NARRATIVE RESPONSES.

The following provides the unedited text of narrative responses from the survey, as referenced in section 5.1.3.

1. The major issues regarding use of programmable logic tools are:
  - Ease of verification
  - Three issues- quality of tool, your design, model used to verify your design.
  - Simulation wrong, wrong timing analysis.
  - No ability to lock down place route and make small changes. Any changes requires full verification suite.
  - Becoming familiarized with the specific CPLD/FPGA vendors fitting tools to synthesize, place and route and program.
  - Conformity of libraries of VHDL and VTL models. Compliance of tool to published standards. (e.g. IEEE VHDL standards).
  - Some people/organizations become complacent and trust the tools to much, then expend their use of a tool without verifying their usage.
  - Failure to identify limitations of verification tools to the design assurance process. For example, simulation tools do not adequately account for electronics effects of real hardware. On the other hand, using gate level simulation is inappropriate for comprehensive timing analysis.
  - Speed. Stability of application of long simulations.
  - Version control (configuration management & baselines) errata list traceability.
  - They do not and can not eliminate physical testing! (Functional).
  - Verification, defect containment, reuse.
  - Clear visibility and traceability to output results.
  - I don't see any major issues with the tools. Most of these tools are widely in use and do not have a negative safety impact. Mainly the repeatability and verification of the design itself affects safety.
  - Errors in timing models resulting in incorrect timing analysis. Errors in simulation models resulting in missed errors. Errors in place and route tools resulting in logic errors in the silicon. I have not experienced any of these issues

- Complexity limits control over build process which can be very important. Advanced features are hidden when they shouldn't be because they are the ones that can solve your problem.
- The tools are designed for rapid freeform development and not for requirements based methodical design. Fine for commercial applications but not really targeted to high reliability design.
- If it can be qualified per DO-254
- Bugs
- Qualification

2. The major issues regarding qualification of programmable logic tools are:

- Availability of design information
- No guidance on what results documents should contain and how that data is utilized/submitted. Comparing simulation clock by clock against logic analyzer on target is one means of quality
- Lack of guidance in terms of what is acceptable.
- Lack of cooperation from programmable logic tools vendor
- Visibility of processes/signals. Willingness of tool vendors (e.g. Cadence; Simplicity) to openly discuss algorithms for simulation or synthesis and conform those algorithms.
- Availability of tool requirement and verifying data. Our suppliers sometimes have to do this themselves.
- Understanding the modes/features of tool that require qualification and applying appropriate qualification based on them.
- Standardized test suite ran by the tool vendor.
- We perform a crucial qualification of ModelSim using only crucial VHDL constructs. How extensive do we have to be? (Within DO-254 guidelines.).
- Price & vendor availability for support.
- Too much reliance on tools simulation to demonstrate functionality. In every case w/near total reliance on tools has resulted in re-spin (in my experience).

- The guidance on tool qualification is not useful. We qualified a tool by comparing a logic analyzer trace from the actual system with a simulation from the tool. This including propagations delays and functionality. The DO-254 guidance also does not include what should be done with the data, i.e. FAA approval or submittal. What kind of tool errors are we looking for, that would help drive the tool qual plan. What are the goals of a DO-254 tool qualification, what should the documentation look like, and what should be done with the qual data? I assume all companies are doing this differently and we have not seen any increase in safety by doing a qualification. These tools are in use in thousands of designs in the world and the probability of a tool qualification effort finding an error is very low or negligible.
  - Assessment is not a big issue because independent verification of the tools is a natural outcome of the design process. Qualification is much more involved and should be done by the vendor. Tool versions change so rapidly that qualification is impractical at the designer level and possibly the vendor level.
  - Frequent tool updates with big fixes limit the amount of time a tool can be "qualified" due to the fact that most tools are completely replaced by the update.
  - Qualification is basically demonstrating the pedigree of the tool itself, most vendors either do not keep requirements based design and verification information or that information is considered proprietary or trades secret. Thus independent assessment of tool output will continue to be the primary, and sometimes only, methodology available for “qualification”. As long as that fact doesn’t change, there will be very little progress.
  - Verification of hard and soft IP cores.
  - Guidelines that are easy to understand.
  - Templates that we can use.
  - Long process to prove qualification
3. Other issues related to programmable logic tools experience are:
- A missing functionality of LIBERO is the netlist editor for netlist after place and route. Any minor change in the design will result in a complete new place and route operation and then all the time consuming verification steps on the netlist must be redone as a consequence any change in an Actel® design should be considered as major.
  - Trace to verification completed on final implementations.

- Altera has something's called a macrofunctions, a t function used in VHDL code. The compiler calls a generic macro function and it works well. For a D0-178 D project I had to write my own 8-bit adder just to avoid qualifying Altera t functions.
- Our biggest problem occur with the chip vendor's place and route tools (e.g. Xilinx; Actel, etc) because they are trying to minimize tool costs -- they are selling chips, not software. Consequently, they don't have good configuration control of the tools/libraries and often introduce errors with new releases.
- Vendors would like to be able to qualify tools once to provide qualified tools to HW vendors.
- I've heard from FPGA vendors that they are attempting to understand what tool qualification means and consider offering it to their customers. While this is a noble goal, I wonder if they have an appropriate understanding of aircraft safety and design assurance to address qualification.
- I was hoping that tool vendors would become more involved to the point that they could sell qualified tools rather than each user trying to figure out how to qualify the tool.
- Use of System Verilog and assertions seem to fit into flow.

## APPENDIX C—TEST PROCEDURE

The document provides details of the proposed test procedures for the Power Integrity, I/O, and Timing Analyses.

A common configuration, shown in figure C-1, will be used for all the test cases. The test will consist of multiple N-bit counters (aggressor signals) surrounding a known (victim) signal. The counters will be synthesized to operate at the maximum frequency allowed by the hardware. All the aggressor and victim input/output (I/O) pins will be assigned to a single bank of the field-programmable gate array (FPGA) and will share power supply resources. The aggressor signal outputs will be driving their maximum specified load. The victim signal will be assigned I/O pins in the middle of the aggressors to force routing in the middle of the aggressor signals.

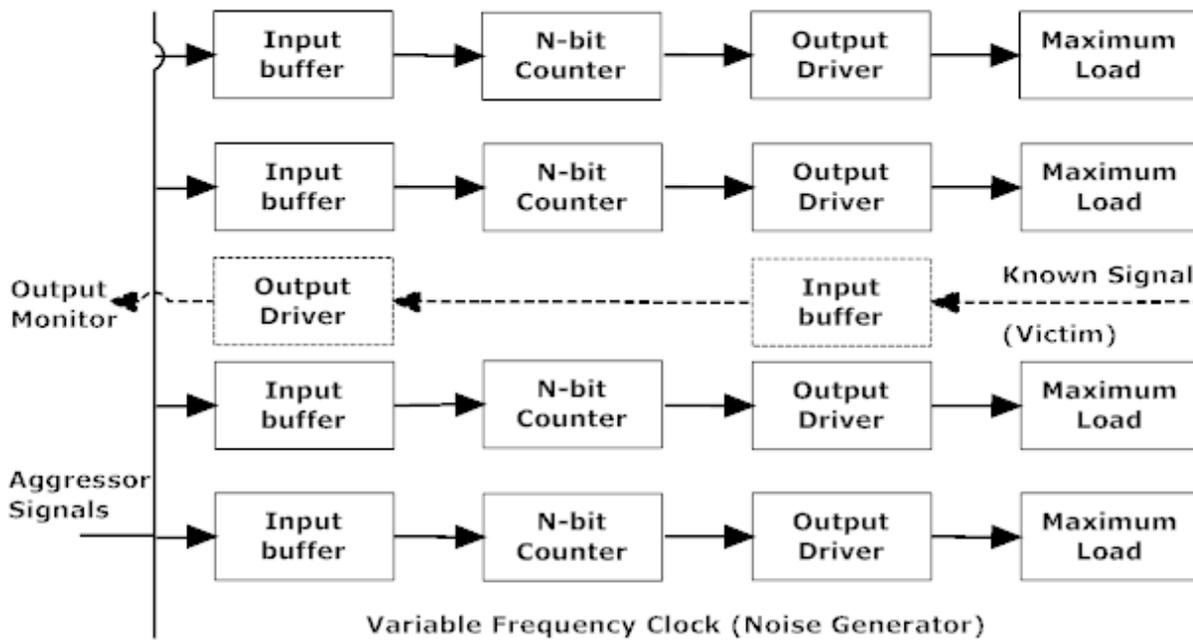


Figure C-1. Ripple Counter

The design will be synthesized and configured to each platform independently. Each platform will require that the timing constraints are set for the ripple counters. It is acceptable that each platform's tightest timing constraints will be different.

### C.1 AIRBORNE ELECTRONIC HARDWARE POWER INTEGRITY ANALYSIS CASE STUDY.

#### C.1.1 OBJECTIVE.

This case study will examine if an airborne electronic hardware (AEH) design tool is aware of potential power integrity issues. The test case will simultaneously switch as many signals on an I/O bank as possible. The signals will be connected to output drivers that will be configured as

LVTTL 24 mA fast drivers. The drivers will be connected to loads intended to maximize the current passing through each I/O. Failures can occur due to excessive inductance in the device power connections or excessive resistive drops in the internal power bussing of the device under test. The test will use one signal on the I/O bank as an input, which will be driven to the maximum voltage for a high input (VIL) (0.8 V) and the minimum voltage for a high input (VIH) (2.0 V) LVTTL specification limits. Failures in the power bussing will be seen as erroneous values read by this input. The outputs will be continuously monitored using an oscilloscope for inconsistencies.

### C.1.2 DESIGN.

This is an overview of how the test will be designed.

- The experiments instruction set will be written so that it is platform-independent.
- All signal lines will be LVTTL (low voltage transistor-transistor logic)
- One input in I/O bank will branch to the remaining outputs in that I/O bank.
- Each I/O pin will be designated as high current and fast slew.

### C.1.3 SETUP.

This is a preparatory procedure for conducting the experiment. This experiment requires a function generator, a radio frequency (RF) generator (>10 GHz), and an oscilloscope. For each platform, it is necessary to create different configuration files to use the development board I/O and save the place-and-route report/configuration.

- The design will be synthesized and configured to each platform independently.
- Each platform will require I/O pin designations be made as high current and fast slew.
- This experiment requires a precision digital power supply, oscilloscope, and logic analyzer.

### C.1.4 DATA COLLECTION PROCEDURE.

This is the data collection procedure:

- During this experiment all the outputs will be monitored by a logic analyzer for logic, or an oscilloscope may be used.
- The input pin, one of the output pins, and the VDD will be monitored with an oscilloscope.

### C.1.5 EXPERIMENT.

The experiment consists of the following steps:

- Connect outputs together.
- Configure and synthesize, and generate and save all tool reports and logs.
- Burn the design onto the board. Save all logs.
- Connect both sources, respectively, and power up all systems. Record the input pin designations.
- Start the data collection procedure.
- Repeat for each platform.

### C.2 AN AEH I/O ANALYSIS CASE STUDY.

#### C.2.1 OBJECTIVE.

The purpose of this case study is to determine the state of undefined I/O pins. The unused FPGA pins, when a component is burned onto the device, could have residual logic, be grounded, be active, or floating. Ultimately, the AEH tool determines what happened to the pins left undefined in a design. The method with which a tool chooses to delegate with these unused pins is a safety concern.

#### C.2.2 DESIGN.

This is an overview of how the test will be designed:

- The code will be a subset of the AEH Power Integrity Analysis code.
- Modify the AEH Power Integrity Analysis code so that the variable frequency signal lines are removed. All the rest of the code remains the same.

#### C.2.3 SETUP.

This is a preparatory procedure for conducting the experiment. This experiment requires a function generator, an RF generator, and an oscilloscope. For each platform, it will be necessary to create different configuration files to use the development board I/O and save the place-and-route report/configuration, including:

- Burning the design to the FPGA immediately following the implementation of the AEH Power Integrity Analysis experiment.

- The design will be synthesized and configured to each platform independently.
- Each platform will require that timing constraints be set for the ripple counters. It is acceptable that each platform's tightest timing constraints will be different.
- All signals will be transistor/transistor logic (TTL).
- Record the timing constraints settings.
- This experiment requires a function generator, an RF generator (>10 GHz), an oscilloscope, and a logic analyzer.
- For each platform, it will be necessary to create different configuration files using the development board I/O's. Replication of the place-and-route report/configuration from AEH Power Integrity Analysis experiment to variable frequency lines will no longer be available for configuration.

#### C.2.4 DATA COLLECTION PROCEDURE.

This is the data collection procedure:

- During this experiment, the varying square wave frequency input will be started at 0 Hz and increased to the RF generator-allowable maximum frequency (>10 GHz) and will be continually monitored for abnormalities, such as wave skew, rounding of wave edges, missed pulses, and non-TTL levels.
- Data will consist of frequency, voltage levels, rise and fall times, noise levels, and interfering frequencies. Data will be collected from all 54 output pins.
- Starting at 0 Hz in logarithmic intervals, data will be recorded. Between the frequency of 950 Hz and 1050 Hz, data points shall be recorded in 10-Hz intervals.
- At each interval, the VDD voltage will be recorded.
- Each of the four fixed signal outputs will be monitored for any second harmonic of the current variable frequency.

#### C.2.5 EXPERIMENT.

The experiment consists of the following steps:

- Burning the design onto an FPGA immediately following the implementation of the AEH Power Integrity Analysis experiment.
- Burning the design onto the platform.

- Use the exact I/O pin designations as they were in the AEH Power Integrity Analysis experiment. Connect the output pins exactly the same with the exception that the current sinking device shall be removed.
- All the previously assigned output pins shall be analyzed with an oscilloscope to determine if they are floating or fixed.
  - Connect the known logic lines to its signal source.
  - Connect the previously assigned variable frequency pin to a 1-kHz signal.
  - Probe all the previously assigned I/O pins, and record their voltage levels for 1 second.
  - Use a 10 k $\Omega$  resistor to pull the previously assigned I/O pins to VDD and measure and record the pin voltage for 1 second.
  - Use a 10 k $\Omega$  resistor to pull the previously assigned I/O pins to ground and measure and record the pin voltage for 1 second.
- Connect the known logic to a 1-kHz-square wave source.
- Execute the Data Collection Procedure. Note: it is acceptable to do a continuous sweep and monitor for any signal. If no signal is found, record the finding. If a signal is found, follow the data collection procedure exactly.

### C.3 AN AEH TIMING ANALYSIS CASE STUDY.

#### C.3.1 OBJECTIVE.

The purpose of this case study is to determine if a tool meets its reported timing constraints and retains a margin of safety. The tools allow the user to specify the time that a specific operation will take to complete. If the bounds of the speed of the gates are pushed closer to their minimums, the tool will redesign the circuit so that the delay is less. The subjects of interest are where these bounds are, how close the tool allows the design to get to the bounds, and whether there is a safety margin with actual delay and estimated delay. During the frequency sweep, both the input and output waves of the component need to be checked for phase accuracy and phase differences on the oscilloscope. As the input frequency increases, the point when either the signal path time is not met or the output wave is not correct is considered a failure. The phase differences will yield the signal path time; the point of failure will determine the safety margins applied. All data and settings should be recorded accurately for analysis.

### C.3.2 DESIGN.

The following is an overview of how the test will be designed:

- This case study will examine a ripple counter. A ripple counter can be instantiated in high-level requirements (HDL) either behaviorally (i.e.,  $A = A + 1$ ) or as a series of flip-flops cascaded together. A 16-bit counter will be coded both behaviorally and explicitly, and will be implemented in both cases.
- The input to the ripple counter will be an external pin that is connected to a high-frequency source. The final output of the counter will be connected to an output pin.
- The output signal frequency will be the input signal frequency over  $2^{16}$ .
- The static timing produced by the tool will be recorded. The hardware will then proceed through place-and-route, and the post place-and-route timing numbers will be used for the test.
- All signals will be LVTTTL.

### C.3.3 SETUP.

This experiment requires a function generator, RF generator, and oscilloscope. For each platform, it will be necessary to create different configuration files to use the development board I/O. The place-and-route report and configuration will need to be saved. The following are the steps that will need to be taken:

- The design will be synthesized and configured to each platform independently.
- Each platform will require that the tool's timing constraints be set for the ripple counters. It is acceptable that each platform's tightest timing constraints will be different.
- The timing constraint settings will be recorded.
- During the initial implementation, a report can be generated by the tool to establish the maximum input frequency and the signal path delay.
- All signals will be TTL.

### C.3.4 DATA COLLECTION PROCEDURE.

The data collection procedure is as follows:

- During this experiment, the varying square wave frequency input will be started at 0 Hz and increased to RF generator-allowable maximum frequency (>10 GHz) and will be

continually monitored for abnormalities, such as glitches, cycle slips, missed pulses, and non-TTL levels. Record these findings.

- Data will consist of frequency, voltage levels, rise and fall times, noise levels, interfering frequencies, and phase difference of input and output waves.
- Starting at 0 Hz and increasing in logarithmic intervals up to the RF generator's maximum frequency, data will be recorded.

### C.3.5 EXPERIMENT.

The implementation will be tested using the following method and will attempt to produce the highest operational frequency for each platform by:

- Synthesizing and burning the design. Generate and save tool reports; make sure to include the timing report and timing constraint settings. Examine the reports to see what optimizations, if any, occurred.
- Connecting the signal generation, and powering up the system.
- Following the data collection procedure.

## APPENDIX D—ANNOTATED BIBLIOGRAPHY

This appendix presents detailed information about design assurance issues that were briefly discussed in section 4 of the main report. The discussions of the research papers shown here are divided into three sections: (1) annotated research papers (section D.1), (2) selection of papers directly related to safety issues detailing the objective and relevance to the project (section D.2), and (3) papers contributed by industry with defined problems and suggested solutions (section D.3).

### D.1 ANNOTATED RESEARCH PAPERS.

The research revealed several papers (i.e., papers 1-26) on the use of software tools in the development of airborne electronic hardware (AEH). This section includes an annotated bibliography of the papers related to general issues of using software tools in electronic hardware development.

1. Aljer, A. and Devienne, P., “Co-Design and Refinement for Safety Critical Systems,” *Proc. DFT '04 19th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, IEEE, 2004, pp. 78-86.

Summary: In this paper, the authors focus on design entry of complex systems, that is, the highest abstract tier of the global system without implementation choices to specific technologies. At this very first level, the use of a formal specification language is considered as the foundation of a real validation process. The paper calls attention to formal design entry, and points out that the project management can be formally controlled by formal refinement. Architecture, based on stepwise refinement of a formal model to achieve controllable implementation, is proposed. This leads to implementations that are highly effective, but remain formally related to the first formal specification. Partitioning, fault tolerance, and system management are seen as particular cases of refinement in order to conceptualize systems that are correct by proven construction. In this paper, the basic principles of system methodologies are presented, and the methodology based on the refinement paradigm is described. To prove this approach, the B-HDL (B hardware description language) tool is based on very high-speed integrated circuits (VHDL) and the B Method (formal language based on set theory and logic) tool has been developed. The benefits of such tools would be an amazing productivity gain, a better reuse automation, and a formal redundancy management.

2. Bannow, N. and Haug, K., “Evaluation of an Object-Oriented Hardware Design Methodology for Automotive Applications,” *Proceedings of the Conference on Design, Automation and Test*, Paris, 16-20 February 2004, Vol. 3, pp. 268-273.

Summary: The authors present results in using the new object-oriented design approach OSSS (ODETTE System Synthesis Subset). The methodology and tools of the ODETTE (tool that uses object-oriented co-design and functional test techniques) project have been developed within the context of the Information Society Technologies program of the

European Commission. The main focus of OSSS lies in the field of hardware design and synthesis capability. The strategy is based on an extension of the synthesizable subset of standard SystemC. The approach supports real object-oriented and synthesizable design features like classes, inheritance, templates, polymorphism, and global object access. Therefore, OSSS promises high efficiency in its capability to handle complex designs, faster development time, improved code quality, and faster time to market. In contrast, standard SystemC is also based on C++ constructs, but no object-oriented constructs are available yet for a synthesizable system description. The OSSS has been evaluated on an automotive design example. It was chosen for the implementation of a component that is part of all video projects: a camera's exposure control unit (ExpoCU). The first main goal that was achieved is a synthesizable design by the automatic generation of a field-programmable gate array (FPGA) netlist from an OSSS description. Furthermore, the methodology seems to be proven to fulfill industrial requirements, such as usability for complex system development, integration of the existing intellectual property (IP), improved code quality, and decreased development effort. Comparison will be done against existing VHDL-based design flow. The paper focuses on implementation and testability by comparing the new object-oriented synthesis approach with a standard VHDL flow by laying emphasis on synthesizability. The OSSS and equivalent methodologies show a potential to handle new generations of complex hardware/software systems. Moreover, the gap between increasing design complexity and available methodologies has become larger and, thus, needs to be closed by new solutions such as OSSS.

3. Bhatt, D., et al., "Model-Based Development and the Implications to Design Assurance and Certification," *Proc. DASC 2005, 24th Digital Avionics Systems Conference*, 30 October - 3 November 2005.

Summary: The term model-based design/development (MBD) has grown in popularity over the past decade. Within the avionics community, the term MBD implies the development and application of "control models and simulations" using tools such as MATLAB/Simulink. At Honeywell, the authors have been engaged in MBD and development of associated tools for avionics applications. This position paper applies the lessons learned and discusses several issues, relating to sound MBD, to meet design assurance and certification objectives. The paper examines the dominant approaches describing commercially available code generation and verification tool suites. The paper contrasts these approaches to traditional software design, implementation, and verification methods. This paper also recommends taking a broader perspective of MBD and suggests adopting lessons learned from the classical software engineering arena considering future investigation, standardization, automation tool development, and integration.

4. Bunker, A., Gopalakrishnan, G., and McKee, S.A., "Formal Hardware Specification Languages for Protocol Compliance Verification," *ACM Trans. on Design Automation of Electronic Systems*, Vol. 9, No. 1, January 2004.

Summary: The advent of the system-on-chip (SoC) and IP hardware design paradigms makes protocol compliance verification increasingly important to the success of a project. One of the central tools in any verification project is the modeling language, and the paper describes the survey of the field of candidate languages for protocol compliance verification, limiting discussion to languages originally intended for hardware and software design and verification activities. The comparison is framed by first constructing taxonomy of these languages, and then by discussing the applicability of each approach to the compliance verification problem. Each discussion includes a summary of the development of the language, an evaluation of the language's utility for the problem domain and, where feasible, an example of how the language might be used to specify hardware protocols. Finally, some general observations are made regarding the considered languages.

5. Bunker A., McKee, S.A., and Gopalakrishnan, G., "An Overview of Formal Hardware Specification Languages," *Grace Hopper Celebration of Women in Computing*, 2002.

Summary: Verification is widely recognized as one of the most difficult aspects of computer hardware design. The gap between design and verification capabilities grows, as does the cost of missed flaws. Many researchers investigate ways to formally verify processor designs, interconnects, and protocols, but creating verification methods and tools will remain a central problem for computer scientists for at least the next decade. This field is explored in the paper by surveying formal specification languages. A taxonomy of languages is presented, and the paper discusses the applicability of each language to standard compliance verification, demonstrating that a hardware design complies to an interconnect standard.

6. Camposano, R. and Wilberg, J., "Embedded System Design," *Design Automation for Embedded Systems*, Vol. 1, No. 1-2, January 1996, pp. 5-50.

Summary: In the past decade, the main engine of electronic design automation (EDA) has been the widespread application of ASIC. Present technology supports complete SoC, most often used as so-called embedded systems in an increasing number of applications. Embedded systems pose new design challenges that will be the driving forces of design automation in the years to come. These include the design of electronic systems hardware, embedded software, and hardware/software co-design. This paper explores novel technical challenges in embedded system design and presents experiences and results of the work in this area using the CASTLE system. CASTLE supports the design of complex embedded systems and the design of the required tools. It provides a central design representation, Verilog<sup>®</sup>, VHDL, and C/C++ front ends, hardware generation in VHDL and Berkeley Logical Interchange Format, a re-targetable compiler backend, and several analysis and visualization tools. Two design examples, video compression and a diesel injection control, illustrate the presented concepts.

7. Chee, W.L., Zain, Ali, N.B., and Nair, R.S., “Design of Low-Cost FPGA-Based PCI Bus Sniffer,” *Proc. FPT 2003 IEEE International Conference on Field-Programmable Technology*, Tokyo, 15-17 December 2003.

Summary: This paper describes FPGA design and implementation of the Peripheral Component Interconnect (PCI) Bus Sniffer—a device used by semiconductor industries to analyze the characteristics of signals transmitted in the PCI bus. These devices are expensive and not easily affordable by individual users. The paper presents a novel method of device design using available freeware tools that facilitate learning hardware design at a low cost. The objective of the paper is to present design and implementation of a low-cost PCI Bus Sniffer using FPGA and the Verilog HDL. The target FPGA device is a Xilinx SPARTAN II with the necessary interface to probe PCI signals. This project has successfully shown that it is possible to design and implement a complex hardware design using freeware tools.

8. Cooper, P.A., “Lessons Learned Using Software-Assisted Systems Engineering on Large Satellite Development Contracts,” *IEEE Aerospace and Electronic Systems Magazine*, Vol. 21, No. 5, May 2006, pp. 7-11.

Summary: Over the years, the world’s defense industries have become quite proficient at developing large, complex hardware and software systems. In recent years, the ubiquitousness of personal computers has changed the way people work and has had a major impact on major systems development efforts. The government’s faster-better-cheaper acquisition philosophy has started driving contractors to a concurrent engineering approach toward systems engineering. This confluence of experts has had unexpected impacts on both the flexibility and rigor of requirements management processes. While the maturing requirements and design hold promise in maintaining requirements traceability throughout the design process, the widespread use of desktop computing systems has inadvertently lulled many experienced systems engineers into sloppy processes because it appears to be a simple matter to make a requirements change in a soft copy of a requirements document. Without strong process and management support, requirements changes may be done in incompatible formats. This author describes the design phase of a major classified government satellite development effort. As an integral member of an extremely experienced requirements management team (boasting over 150 years of combined experience in the defense industry), the author had the opportunity to watch the team navigate straight into many of the systems engineering potholes created when talented engineers implement concurrent engineering using a variety of tools without a consistent process framework. This paper, therefore, specifically addresses process and implementation challenges that arose when establishing a software-assisted, concurrent-engineering project.

9. Dajani-Brown, S., Cofer, and D.’ Bouali, A., “Formal Verification of an Avionics Sensor Voter Using SCADE,” *Proc. FORMATS 2004 Joint International Conference on Formal Modelling and Analysis of Timed Systems, and FTRTFT 2004 Formal Techniques in*

*Real-Time and Fault-Tolerant Systems*, Lecture Notes in Computer Science, Vol. 3253, pp. 5-20.

Summary: Redundancy management is widely used in mission-critical digital flight control systems. This study focuses on the use of Safety-Critical Application Development Environment (SCADE) and its formal verification component, the Design Verifier, to assess the design correctness of a sensor voter algorithm used for management of three redundant sensors. The sensor voter algorithm is representative of embedded software used in many aircraft today. The algorithm, captured as a Simulink diagram, takes input from three sensors and computes an output signal and a hardware flag, indicating correctness of the output. This study is part of an overall effort to compare several model checking tools to the same problem. SCADE is used to analyze the voter's correctness in this part of the study. Since synthesis of a correct environment for analysis of the voter's normal and off-normal behavior is a key factor when applying formal verification tools, this paper is focused on (1) the different approaches used for modeling the voter's environment and (2) the strengths and shortcomings of such approaches when applied to the problem under investigation.

10. Hayek, A. and Robach, C., "From Specification Validation to Hardware Testing: A Unified Method," *Proc. International Test Conference*, 20-25 October 1996, pp. 885-893.

Summary: With the advancements in the design automation field, tools make it possible to describe hardware systems as software programs using high-level HDLs, such as VHDL or Verilog. Consequently, a design fault that affects the system specification can be considered a software fault. To test the system specification against (software) design faults, the authors propose an adaptation of a mutation analysis, originally proposed for software testing, to test VHDL functional description. The resulting test set is applied to the gate-level structure of the system to measure its capacity to uncover hardware faults, such as the stuck-at faults. Heuristics are presented to enhance the test set in order to be sufficient for testing hardware faults, and the results are compared to traditional automatic test pattern generation. Accordingly, this paper presents a unified method for testing both the system specification and the hardware implementation.

11. Hilton, A.J., "High-Integrity Hardware-Software Codesign," Ph.D. Thesis, The Open University, Milton Keynes, United Kingdom, April 2004.

Summary: Programmable logic devices (PLD) are increasing in complexity and speed and are being widely used in safety-critical systems. Methods for developing high-integrity software for these systems are well known, but this may not be true for programmable logic. The author proposes a process for developing a system incorporating both software and PLD, suitable for safety-critical systems of the highest levels of integrity. This process incorporates the use of Synchronous Receptive Process Theory (SRPT) as a semantic basis for specifying and proving properties of programs executing on PLD. This process also extends the use of the Southampton Program Analysis Development Environment Ada Kernel (SPARK Ada) from a programming

language for safety-critical systems software to cover the interface between software and programmable logic. The proposed approach has been validated through the specification and development of a substantial safety-critical system incorporating both software and programmable logic components and the development of tools to support this work. The authors claim that the demonstrated methods are not only feasible but also scale up to realistic system sizes, allowing development of such safety-critical software and hardware systems to the levels required by current system safety standards.

12. Hilton, A. and Hill, J., "On Applying Software Development Best Practices to FPGAs in Safety-Critical Systems," *Field Programmable Logic and Applications: The Roadmap to Reconfigurable Computing*, 10th International Conference, FPL 2000 Villach, Austria, August 27-30, 2000 Proceedings, Lecture Notes in Computer Science, Volume 1896, pp. 793-796, Springer Berlin Heidelberg, 2000.

Summary: New standards for developing safety-critical systems require the developer to demonstrate the safety and correctness of the programmable logic in such systems. The paper describes adaptation of software development best practices to developing high-integrity FPGA programs.

13. Hilton, A.J. and Hall, J.G., "Developing Critical Systems with PLD Components," *Proc. FMCIS 10th International Workshop on Formal Methods for Industrial Critical Systems*, September 2005.

Summary: Understanding the roles that rigor and formality can have in the design of critical systems is very important in their development. Whereas software developers have sufficient knowledge of these issues, for the developers of PLDs, and specifically for the combination of PLDs and software, the issues are less known. Indeed, even in industry there are differences between current and recommended practice, and engineering opinions differ on how to apply existing standards. This situation has led to gaps in the formal and rigorous treatment of PLDs in critical systems. In this paper, the range of, and potential for, formal specification and analysis techniques that address the requirements for verifiable PLD programs are examined. The existing formalisms that may be used are identified. The areas of contributions that academia and industry could make collaboratively would allow high-integrity PLD programming to be as practicable as high-integrity software development. The paper briefly discusses important practical, technical, organizational, social, and psychological aspects of the introduction of formal methods into industrial practice for hardware and system design. It also provides an update and summary of the recent UK Defence Standard 00-56, as it relates to hardware.

14. Hilton, A.J., Townson, G., and Hall, J.G., "FPGAs in Critical Hardware/Software Systems," *Proc. FPGA 2003 ACM/SIGDA 11th International Symposium on Field Programmable Gate Arrays*, Monterey, CA, ACM, 2003, p. 244.

Summary: FPGAs are being used in increasingly complex roles in critical systems, interacting with conventional critical software. Established safety standards require

rigorous justification of safety and correctness of the conventional software in such systems. Newer standards now make similar requirements for safety-related electronic hardware, such as FPGAs, in these systems. In the paper, the current state-of-the-art in programming FPGAs, and their use in conventional (low-criticality) hardware and software systems are examined. The paper discusses the impact that the safety standards requirements have on the co-development of hardware/software combinations in critical systems and suggests adaptations of the existing best practice in software development that could discharge them. Particular attention is paid to the development and analysis of high-level language programs for FPGAs designed to interact with conventional software.

15. Hoskote, Y.V., et al., “Automatic Verification of Implementations of Large Circuits Against HDL Specifications,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, March 1997.

Summary: This paper addresses the problem of verifying the correctness of gate-level implementations of large, synchronous, sequential circuits with respect to their higher-level specifications in HDL. The verification strategy is to verify containment of the finite state machine (FSM) represented by the HDL description in the gate-level FSM by computing pairs of compatible states. This formulation of the verification problem dissociates the verification process from the specification of initial states, whose encoding may be unknown or obscured during optimization and also enables verification of reset circuitry. To make verification of large circuits with merged data path and control tractable, the concept of strong containment is introduced. This is a conservative approach that exploits correspondence between data path registers in the two descriptions without requiring any correspondence between the control units. An important result with associated proof that computation of pairs of equivalent, or compatible, states can be achieved by considering subsets of the circuit outputs is presented. Consequently, verification of circuits with large and diverse I/O sets, which were previously intractable, due to lack of a single effective variable order for the binary decision diagrams, are now feasible. Experimental results are presented for the verification of several industry level circuits.

16. Karlsson, K. and Forsberg, H., “Emerging Verification Methods for Complex Hardware in Avionics,” *Proc. DASC 2005, 24th Digital Avionics Systems Conference*, 30 October—3 November 2005, Vol. 1, pp. 6.B.1-61-12.

Summary: This paper discusses the additional design assurance strategies stated in RTCA DO-254, appendix B, “Design Assurance Considerations for Level A and Level B functions.” In particular, the use of formal specification languages, such as the property specification language in combination with dynamic (simulation) and static (formal) verification methods for PLDs, are addressed. Using these methods, a design assurance strategy for complex programmable airborne electronics compliant with the guidelines of DO-254 is suggested. The proposed strategy is a semi-formal solution, a hybrid of static and dynamic assertion-based verification.

17. Kern, C. and Greenstreet, M.R., "Formal Verification in Hardware Design: A Survey," *ACM Transactions on Design Automation of Electronic Systems*, Vol. 4, No. 2, pp. 123-193, 1999.

Summary: In recent years, formal methods have emerged as an alternative approach to ensuring the quality and correctness of hardware designs, overcoming some of the limitations of traditional validation techniques, such as simulation and testing. There are two main aspects to the application of formal methods in a design process: (1) the formal framework used to specify desired properties of a design and (2) the verification techniques and tools used to reason about the relationship between a specification and a corresponding implementation. The paper presents a survey of a variety of frameworks and techniques proposed in the literature and applied to actual designs. The specification frameworks include temporal logics, predicate logic, abstraction and refinement, as well as containment between regular languages. The verification techniques presented include model checking, automata-theoretic techniques, automated theorem proving, and approaches that integrate the above methods. To provide insight into the scope and limitations of currently available techniques, a selection of case studies where formal methods were applied to industrial-scale designs, such as microprocessors, floating-point hardware, protocols, memory subsystems, and communications hardware, are presented.

18. Marcon, C.A.M., et al., "Prototyping of Embedded Digital Systems From SDL Language: A Case Study," *Proc. HLDVT'02 Seventh IEEE International High-Level Design Validation and Test Workshop, Cannes, France, 27-29 October 2002*, pp. 133-138.

Summary: The author's goal was to evaluate the performance of embedded digital systems generated from a system-level description language. The target language is SDL, which is automatically synthesized with a co-design tool, resulting in VHDL and C descriptions. The co-design tool is responsible for software, hardware, and communication synthesis. Two case studies are presented, exploring the results with respect of the chip area and delays. The results focus on the hardware synthesis, since the goal is to compare the performance of systems generated from a hand-coded HDL descriptions against a synthesized HDL. The analysis of the advantages and drawbacks of this automatic hardware design flow and the evaluation of the commercial tools integration are also reported.

19. Mencer, O., "ASC: A Stream Compiler for Computing With FPGAs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 25, No. 9, September 2006, pp. 1603-17.

Summary: A Stream Compiler (ASC) for computing with FPGA emerges from the ambition to bridge the hardware-design productivity gap where the number of available transistors grows more rapidly than the productivity of very large-scale integration and FPGA computer-coded design tools. ASC addresses this problem with a software-like programming interface to hardware design while maintaining the performance of hand-designed circuits. ASC improves productivity by letting the programmer optimize the

implementation on the algorithm, architecture, arithmetic, and gate levels, all within the same C++ program. The increased productivity of ASC is applied to the hardware acceleration of a wide range of applications. Traditionally, hardware accelerators are tediously handcrafted to achieve top performance. ASC simplifies design space exploration of hardware accelerators by transforming the hardware design task into a software design process, using only the “make” process to obtain a hardware netlist. From experience, the hardware design productivity and ease of use are close to pure software development. This paper presents results and case studies with three levels of optimizations: (1) on the gate level—Kasumi and International Data Encryption Algorithm encryptions, (2) on the arithmetic level—redundant addition and multiplication function evaluation for two-dimensional rotation, and (3) on the architecture level—Wavelet and Lempel-Ziv (LZ)-like compression.

20. Mills, M. and Peterson, G., “Hardware/Software Co-Design: VHDL and Ada 95 Code Migration and Integrated Analysis,” *Proc. 1998 Annual ACM SIGAda International Conference on Ada*, Washington, DC, 1998, pp. 18-27.

Summary: Optimizing the design is an important task in efficiently developing and deploying effective complex weapons systems. Often, architectural tradeoffs between hardware and software implementation must be performed early in the design cycle, resulting in potentially inefficient systems or subsystems. As technologies and costs in hardware and software implementation change over time, the optimal partitioning of system functionality into hardware and software components may also change. Currently, recasting a component from hardware to software (or vice-versa) is a difficult and error-prone activity. The paper explores a new approach to ease the hardware/software co-design and repartitioning activities by providing a mechanism to exchange software written in Ada 95 with behavioral VHDL.

21. Miner, P.S., et al., “A Case-Study Application of RTCA DO-254: Design Assurance Guidance for Airborne Electronic Hardware,” *Proc. DASC 2000, 19th Digital Avionics Systems Conferences*, Vol. 1, pp. 1A1/1-1A1/8.

Summary: In a joint project with the Federal Aviation Association (FAA), National Aeronautics and Space Administration (NASA) Langley Research Center is developing a hardware design in accordance with DO-254. The purpose of the case study was to gain understanding of the guidance document and generate an example suitable for use in training. For the case study, a core subsystem of the Scalable Processor-Independent Design for Electromagnetic Resilience, which is a new fault-tolerant architecture under development at NASA Langley Research Center, was selected.

22. Nehme, C. and Lundqvist, K., “A Tool for Translating VHDL to Finite State Machines,” *Proc. 22nd Digital Avionics Systems Conference*, Vol. 1, pp. 3.B.6-1-7.

Summary: The paper describes a framework for design, verification, and execution of safety-critical applications. The framework consists of both software tools for application

verification and hardware platforms for execution and real-time monitoring. The paper discusses the development of a tool to translate safety-critical VHDL code into a formal representation in a form of an FSM model. Different formal techniques can then be applied on this representation to verify properties, such as liveness and deadlock, and to validate that the timing constraints of the original system hold. This paper will discuss three aspects of the tool implementation: transformation of source code into an intermediate representation, verification of real-time properties, and some tool-related implementation issues.

23. Peterson, G.D. and Hines, J.W., "Advanced Avionics System Development: Achieving Systems Superiority Through Design Automation," *Proc. 1998 IEEE Aerospace Conference*, 1998. Vol. 1, Issue 21, pp. 231-238.

Summary: Avionics systems in advanced aircraft provide the improved capability critical to achieving mission success for the war fighter. As the costs associated with aircraft avionics continue to mount, improved weapons system acquisition and support depends on cost-effective design methodologies and accurate design documentation. This paper explores how the standard hardware description language VHDL serves a critical role in effective acquisition of digital electronic systems. Wright Laboratory programs focusing on electronic systems design automation provide complementary improvements in design, documentation, and maintenance capabilities. Results from this research supports acquisition reform efforts to streamline the weapons system procurement process and provide contractors the flexibility to use the most effective design management techniques. At the same time, while the U.S. Department of Defense is moving away from dictating standards in contracting, the electronics industry continues to embrace open standards as a means to ensure hardware and software component compatibility. The question arises: what methodology and standards developments are necessary to support the continuing development of sophisticated weapons systems for the military? To address this question, the paper explores methodological needs for hardware and software design, manufacturability, test, and related issues to provide context and motivation before describing ongoing work to meet these needs.

24. Salzwedel, H., "Mission Level Design of Avionics," *Proc. 23rd Digital Avionics Systems Conference*, Vol. 2, pp. 9.D.2-1-10.

Summary: Aerospace systems are characterized by architectural complexity, dynamic interaction between subsystems, and complex functionality, which are understood by teams from different disciplines. Twenty years ago, the major challenge was the multidisciplinary design of avionics. Over the past 20 years, design methods and tools have been developed to cope with these challenges. Today, the complexity of networked electronics in aircraft and the interaction of hardware and software impose similar complexity and design challenges. The complexity of electronics, according to Moore's law, closely followed by industry, increases by a factor of 100 every 10 years. To cope with this increase of complexity, an increasing abstraction in the design methodology is

required. This paper shows the move towards performance and mission-level design and its advantages over functional-level design approaches.

25. Sangiovanni-Vincentelli, A. "Quo, Vadis, SLD? Reasoning About the Trends and Challenges of System-Level Design," *Proceedings of the IEEE*, Vol. 95, No. 3, March 2007, pp. 467-506.

Summary: The paper discusses system-level design (SLD) considered by many as the next frontier in EDA. SLD means many things to different people, since there is no consensus on a definition of the term. Academia, designers, and EDA experts have taken different avenues to attack the problem, for the most part, springing from the basis of traditional EDA and trying to raise the level of abstraction at which IC designs are captured, analyzed, and synthesized from. However, this is just the tip of the iceberg of a much larger problem that is common to all system approach practitioners. In particular, notwithstanding the obvious differences among industrial segments (for example, consumer, automotive, computing, and communication), there is a common underlying basis that can be explored. This basis may yield a novel EDA industry and even a novel engineering field that could bring substantial productivity gains, not only to the semiconductor industry, but to all system approach practitioners, including industrial and automotive, communication and computing, avionics and building automation, space and agriculture, and health and security, in short, a real technical renaissance. The paper presents the challenges faced by industry in SLD. A design methodology, platform-based design (PBD), that has the potential of addressing these challenges in a unified way is proposed. The methodology and tools available today in the PBD framework and a tool environment called Metropolis that both supports PBD and can be used to integrate available tools and methods together with two examples of its application to separate industrial domains are presented.

26. Turner, K.J. and He, J., "Formally Based Design Evaluation," *Proc. CHARME 2001, Proc. 11th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods, Lecture Notes in Computer Science*, Vol. 2144, pp. 104-109.

Summary: The paper investigates specification, verification, and test generation for synchronous and asynchronous circuits. The approach is called digital logic in LOTOS, the International Organization for Standardization language of temporal-ordering specification, or DILL for short. Relations for strong conformance are defined to verify a design specification against a high-level specification. Tools have been developed for automated testing and verification of conformance between an implementation and its specification.

## D.2 SAFETY ISSUES.

Several of the papers listed in section D.1 also address specific issues of safety in AEH tools (i.e., papers 13, 14, 16, 21, 23, and 24). The selected literature entries below were analyzed in-depth

to identify the specific paper objective as related to the safety issue, provide more extensive but brief description, and specify the relevance to the project. The reference numbers from section D.1 were retained in this section to facilitate comparison with those same papers from the perspective of safety issues.

13. Hilton, A.J. and Hall, J.G., “Developing Critical Systems With PLD Components,” *FMCIS 10th International Workshop on Formal Methods for Industrial Critical Systems*, September 2005.

**Paper Objective:** The paper provides recommendations on guidelines for PLDs used in safety-critical systems. PLDs constitute increasingly important components of safety-critical systems. By placing specific processing tasks within auxiliary hardware, the software load on a conventional central processing unit (CPU) can be reduced, leading to improved system performance. They are also used to implement safety-specific functions that must be outside the direct address space of the main CPU.

**Brief Description:** The paper addresses high-integrity requirements of contemporary AEH. DO-254 specifies the verification recommended for component testing based on a functional failure path analysis (FFPA) that decomposes the identified hazards related to the component into safety-related requirements for the design elements of the hardware program. The verification that DO-254 suggests may include some or all of the following:

- Architectural mitigation: Changing the design to prevent, detect, and correct hazardous conditions.
- Product service experience: Arguing reliability based on the operational history of the component.
- Elemental analysis: Applying detailed testing and/or manual analysis of safety-related design elements and their interconnections.
- Safety-specific analysis: Relating the results of the FFPA to safety conditions on individual design elements and verifying that these conditions are not violated.
- Formal methods: The application of rigorous notations and techniques to specify or analyze some or all of the design.

The common requirements for safety-critical devices DO-254 are the following:

- To operate under an appropriate quality/safety management system
- To plan the development process and the safety argument in advance
- To consider both random and systematic failures

- To qualify tools involved directly in the compilation chain
- To use analytic techniques (e.g., formal methods) to verify high-integrity programs
- To conduct the verification based on identified system hazards

The paper highlights the cases where PLDs were used in a critical function for a system and related safety concerns. An example is using an FPGA in a space-based tethering experiment where an unanticipated power-up characteristic of the chosen FPGA caused the effective loss of the satellite. Despite extensive testing, it was not possible to reproduce the transient spike twice within several hours—a classic transient fault. It is clear from this analysis that extensive testing is not sufficient for mission or safety-critical FPGAs; it is equally true that even formal analysis and proof would be unlikely to detect such a problem.

The approach to using FPGAs in a hostile environment is described. The main environmental hazard in their target domain (space) is corruption of volatile memory via bombardment by high-energy particles. The architecture adopted is a triple-redundant design with fault detection and periodic “scrubbing” to reset all look-up tables to known values. This is a classic example of mitigating an unavoidable hazard; however, the design and function greatly complicates the task of arguing system correctness. The user must balance increased general reliability against demonstrable correctness.

Recent research and development has contributed to the problem of producing high-integrity PLD programs. Research relevant to safety-critical PLD program design includes the following:

- Specification and proof of parallel systems, enabling a correct-by-construction approach to program design
- Model-checking techniques to verify safety properties of an existing PLD design at an HDL or netlist level
- The design and use of high-level programming languages to enable PLD programming at a more abstract level, possibly in a domain-specific language or tool

Relevance to the project: This paper focuses on the DO-254 approach for verification and integration. It also emphasizes that because of the insufficient product service experience of the VHDL compiler to qualify the tool according to DO-254 requirements, the synthesized output was inspected manually to ensure that the critical design components were present and correctly connected.

14. Hilton, A.J., Townson, G., and Hall, J.G., "FPGAs in Critical Hardware/Software Systems," *Proc. FPGA 2003 ACM/SIGDA 11th International Symposium on Field Programmable Gate Arrays*, Monterey, CA, ACM, 2003, pp. 244.

Paper Objective: The paper discusses safety-critical systems implemented in FPGA technology. A critical system is one in which failure may lead to serious consequences for the operator. Failure of safety-critical systems may endanger human life; failure of business-critical systems may result in serious financial loss. In such systems, preventing failure is crucial.

Brief Description: There are possible complications brought on by using FPGAs, e.g., the highly parallel nature of their computations, the difficulties of interfacing to other system components, and timing issues. Timing issues can be resolved (to some extent) by using a simulation or a synchronous HDL. Interfacing problems can be addressed by implementing an asynchronous interface protocol. However, correct analysis of the parallel structure of FPGAs remains the key to extending software development best practice to the PLD domain.

Relevance to the Project: The paper highlights the concerns related to FPGAs: the highly parallel nature of their computations, the difficulties of interfacing to other system components, and timing issues that may impact safety, which are further examined in the project case studies.

16. Karlsson, K. and Forsberg, H., "Emerging Verification Methods for Complex Hardware in Avionics," *Proc. DASC 2005, 24th Digital Avionics Systems Conference*, 30 October - 3 November 2005, Vol. 1, pp. 6.B.1 - 61-12.

Paper Objective: During the last 10 years, there has been a tremendous increase in the ability to design AEH. However, the ability to verify correctness of complex hardware has increased at the same pace. Since verification and testing represents the larger part of development expenses, the industry needs to tackle a test/cost problem, which is particularly true for the highest design assurance (levels A and B) in airborne systems. In particular, the paper addresses the use of formal specification languages, such as the property specification language in combination with dynamic (simulation) and static (formal) verification methods for PLD. Using these methods, the author suggests a design assurance strategy for complex, programmable airborne electronics compliant with DO-254. The strategy is a semiformal solution, a hybrid of static and dynamic assertion-based verification.

Brief Description: This paper highlights one safety concern for an FPGA/PLD by identifying accepted architectural mitigation techniques: (1) TRM and (2) hardware diversity. The concept of TRM mitigates single-event upsets in the FPGA registers. Hardware diversity using two different technology-independent FPGAs might be used if one cannot prove or assure that the FPGA component itself is free of design faults (or if the FPGAs place-and-route tool cannot be shown to be reliable and the output is not

independently assessed). FPGA hardware diversity, however, increases both the cost and the complexity of the design and should therefore be avoided.

This paper particularly addresses formal methods to be used in the FPGA/PLD design/verification flow for hardware design assurance levels (DAL) A, B, and C in airborne applications. The functional specification can be used for both documentation of requirements and verification of the design's compliance. It is possible to tightly connect documents and reviews to present a complete and consistent design/verification flow.

Relevance to the Project: The paper helps one to understand how to develop more confidence in the design and to reduce time spent on exhaustive simulation, which also reduces the risk of discovering bugs in late stages of the design and provides adequate means of measuring and presenting functional/structural coverage for safety-critical functions.

21. Miner, P.S., et al., "A Case Study Application of RTCA DO-254: Design Assurance Guidance for Airborne Electronic Hardware," Digital Avionics Systems Conferences, 2000, *The 19th Proceedings DASC*, Vol. 1, Issue 2000, pp. 1A1/1-1A1/8.

Paper Objective: In a joint project with the FAA, NASA Langley Research Center is developing a hardware design in accordance with DO-254. The purpose of the case study is to understand the new guidance document and generate an example suitable for use in training.

Brief Description: This document is intended to provide a basis for the certification of AEH devices used in aircraft. NASA Langley Research Center also has a secondary objective for this case study: to develop a hardware platform supporting in-house research targeted toward demonstrating systematic recovery from multiple correlated transient failures.

For the case study, the authors have chosen to design a central subsystem of a new fault-tolerant architecture, emphasizing the role of early life cycle in the subsequent verification activities. The correctness of the conceptual design is the prerequisite to assure correctness of the detailed design and implementation.

The principal focus of conceptual design verification activities is formal proof that the fault tolerance protocols are correct. Subsequent design and verification activities will be focused on preserving the implementation integrity of the verified algorithms.

Relevance to the Project: The paper helps the reader understand the role of early conceptual design and verification activities and their impact on the success of safety-related design and development.

23. Peterson, G.D. and Hines, J.W., "Advanced Avionics System Development: Achieving Systems Superiority Through Design Automation," *Proceedings of the 1998 IEEE Aerospace Conference*, 1998. Vol. 1, Issue 21, pp. 231-238.

Paper Objective: Conservative estimates predict high-performance chips exceeding 100-million logic transistors and 1-GHz speeds that will be commercially available, thus rendering a strategy of achieving productivity improvement, which is impractical for all but the largest and most critical design efforts. Decreasing feature size for devices creates further demands because analog, electromagnetic, and atomic effects become more significant in submicron design, thus invalidating assumptions and models previously employed.

Brief Description: The pace of technological innovation and the competitiveness in the commercial market results in continuous reduction of product lifetimes for commercial parts and manufacturing processes.

Design methodologies and tools must ensure correct and efficient implementations of designs. Simply exploiting improvements in designing and manufacturing electronics hardware solves only half the problem: efficient and reliable methods are needed to develop and maintain both the hardware and software. With advances in reconfigurable computing, the delineation between these two domains continues to blur. Effective language support, tools, and methodologies addressing decreasing device size and higher speeds will help enable the future deployment of sophisticated systems and the affordable, effective maintenance of existing weapons systems.

To be able to adequately support a methodology for designing complex weapons systems, accurate specifications of function, timing, interface, and constraints are needed.

During integration and test, performance bottlenecks and hardware bugs are discovered. Due to the high cost of redesigning hardware, these problems are typically rectified by making modifications to the system software, resulting in code that is late and over budget. At this point, additional software engineers may be added to the effort, which can exacerbate the schedule and budget problems. The software development effort often receives the blame for the overall project difficulties when, in reality, the problems come from communication and coordination shortcomings during specification and development.

Relevance to the Project: Often there is little interaction between the hardware and software design efforts due to lack of a unified representation, simulation, and synthesis framework. Because the integration and test phase of the design process is typically the first time the hardware and software are joined, a variety of problems are often encountered. Often, changes in the hardware design are not communicated to the software design team, so the software is developed for the wrong hardware configuration.

24. Salzwedel, H., "Mission Level Design of Avionics," *Proc. 23rd Digital Avionics Systems Conference*, Vol. 2, pp. 9 D.2-1-10.

**Paper Objective:** Aerospace systems are characterized by architectural complexity, dynamic interaction between subsystems, and complex functionality. Today, the complexity of networked electronics in aircraft and the interaction of hardware and software impose similar complexity and design challenges. This paper shows the move toward performance- and mission-level design and its advantages over functional-level design approaches.

**Brief Description:** Electronic chips have become systems and complex systems like aircraft, spacecraft, automobiles, and communication systems are dominated by networked electronics with embedded software. This compounds the problem of the gap between design and implementation. The increase in complexity has caused major problems throughout the industry, including:

- The failure of the first Ariane 5 rocket because of a numerical value overflow. The implementation was not tested against the mission.
- The development of the Teledesic satellite system being discontinued after it was found that major design specifications had to be revised late in the design.
- Two spacecraft to Mars failing in 1999 because of a mixup between units used by different design teams.

Each of the design engineers and design teams makes certain assumptions about how their design will interact with near or far subsystems of other design teams. These assumptions will not be the same, and some will not be documented. Additionally, many subsystems may be reactive with respect to the environment and, thus, events cannot be predicted. Hardware descriptions may be incompatible for those with different subsystems. Insufficient communication between design teams will prevent required information from being passed along. Simulations of the overall functional or implementation models are not possible, and the overall system cannot be validated and verified on a computer.

When the independently developed subsystems are put together to create the overall system, it does not work at first. Problems are fixed on a local level. Validation against overall system requirements cannot be made since they are not executable and may be inconsistent. Distant effects are often discovered late in the design, or worse, during operation by the customer.

The critical problems could be in hardware or software, but more often in the coupling between them. A major contributor to this problem is that the designs are done at the functional level. However, complex systems can no longer be simulated as a whole at functional level.

The following hierarchical mission-level design approach was developed; it generalizes the design approach for deep space missions and makes design decisions where quantitative information is first available:

- A validated and executable mission is the behavior of the system that uses a component to be designed.
- Validated specifications of the functional behavior are generated by validating the high level architecture and performance of the component against the mission level requirements.
- The functional behavior of hardware and software is verified and validated separately and in combination against the specifications stemming from the architectural/performance model.

This paper emphasizes the role of modeling and simulation at the architectural and performance levels that permit the development of executable specifications, significantly reducing the probability of critical design errors and reducing the number of design iterations and hence reducing cost. An integrated design process is described that integrates the design from mission-level requirements to hardware and software implementations and verifications.

Relevance to the Project: Critical issues are standardization of models at the architectural performance level and validation at the architectural and performance levels. The electronic hardware becomes obsolete much faster than software and may not be available for the lifetime of an aerospace system. The issue is: How can hardware be replaced without changing the embedded software?

### D.3 INDUSTRY PROBLEMS AND SOLUTIONS.

Literature entries on tool qualification from an industry perspective were collected as a result of the project research reflecting industry practices for using AEH tools for design and verification of PLDs. The following research papers (i.e., papers 27-45) continue the list begun in section D-1 that ended with paper 26.

27. Lange, M., "Assessing the ModelSim Tool for Use in DO-254 and ED-80 Projects," Rev. 1.1, Mentor Graphics<sup>®</sup> Corp., May 2007.

Identified Problem: Assessment of Mentor Graphics'<sup>®</sup> ModelSim tool according to DO-254. ModelSim is considered a verification tool, performing design analysis so that no design errors are missed. The tool does not modify the design. The tool assessment must be performed to provide confidence to the certification agency that ModelSim is adequate to carry out the verification activity.

Suggested Solution: According to DO-254 (flow diagram for “Design and Verification Tool Assessment and Qualification”), three methods exist for tool assessment:

- Independent output assessment
- Relevant tool history
- Tool qualification

ModelSim is considered a verification tool in the aerospace and avionics certification process. It is used for digital simulation of directed test cases and provides coverage data. The paper identifies the steps and documentation needed for assessing ModelSim in the design process, following the flow diagram from DO-254. The ten steps are described with the author’s interpretation of the necessary activities. Several examples are given from the ModelSim manual and other sources that help identify major issues in going through the qualification flow diagram.

28. Berens, K., “NASA Complex Electronics Guidebook for Assurance Professionals,” December 2004.

Identified Problem: Both software and quality assurance engineers need to understand what airborne electronic devices are, where they are used, and how are they designed. Since detailed assurance guidance is not available, some kind of guidebook is necessary to increase confidence in the quality of airborne electronics.

Suggested Solution: The guidebook provides an introduction to the subject of complex electronics and informs the reader on the following issues:

- Which devices are complex and which are not?
- Overview of airborne electronic devices, including NASA projects using these devices
- How electronics engineers design and program these devices
- Assurance and verification activities for complex electronics
- NASA’s direction regarding assurance activities for airborne electronics

Ultimately, the role of the guidebook is to provide the reader with a general understanding of airborne electronic devices and the design and assurance activities.

29. A380 Certification Review Item F09, Issue 2, “Digital Devices Design Assurance,” March 2003

Identified Problem: Airbus<sup>®</sup> proposed to use PLDs in A380 airborne systems. PLDs are considered to be devices with a complexity that may be equivalent to software. An

industry standard, ED-80 (DO-254), has been issued for the design assurance aspects of AEH. However, it necessitates some clarification when applied to PLD. The purpose of this document is to define specific guidance for certification aspects associated with PLD for systems containing digital electronics on the A380 aircraft.

**Suggested Solution:** The document discusses the Joint Aviation Authority's (JAA) position on compliance with the ED-80 (DO-254) and the corresponding position of Airbus Industries. Among a variety of discussed issues, two positions on tool assessment and qualification are quoted and declared compatible:

- Item 4.2.6, page 4, JAA: "For levels A and B, assurance compliant with the intent of ED-80 should be provided for development and verification tools. A claim for credit of relevant tool history, as discussed in ED-80 Section 11.4.1 item 5, should be justified to the authority."
- Item 4.1.8, page 8, Airbus: "For levels A and B, assurance compliant with DO-254/ED-80 should be provided for development and verification tools. A claim for credit of relevant tool history, as discussed in DO-254/ED-80 Section 11.4.1 item 5, should be justified to the authority."

30. Hilton, A. and Hill, J.G., "On Applying Software Development Best Practices to FPGAs in Safety-Critical Systems," The Open University, 2000.

**Identified Problem:** Standards, such as the UK Defence Standard (Def Stan) 00-54 and IEC 61508, for developing safety-critical systems require the developer to demonstrate the safety and correctness of the PLD in such systems. In addition, programming such devices is similar to programming conventional microprocessors in terms of program size, complexity, and the need to clarify a program's purpose and structure. Def Stan 00-54 includes several recommendations that put emphasis on a formal language to support reasoning about programmable logic behavior to assist developers to comply with this standard. Without the ability to reason formally, it is not possible to meet several requirements of the standard. This is especially true for HDLs without formal semantics, such as the VHDL or Verilog<sup>®</sup>, which are commonly used in hardware design.

**Suggested Solution:** This paper identifies three distinct needs for clear semantics of FPGA programs to be able to

- demonstrate that programs satisfy their specifications.
- refine designs into code while demonstrating their semantic equivalence.
- reason about behavior at the interface between software and programmable logic.

The paper suggests using the SRPT to reason about the FPGA as a collection of small SRPT processes reacting to input signals to produce outputs, when cells are viewed as processes and their routing is viewed as describing which signals pass to which process. The authors suggest refining an FPGA program design from the Z specification language

to an implementation, maintaining demonstrable correctness. A useful stepping stone would be a programming language that could act as the target of refinement from Z and then could be compiled into an SRPT process. One suggested candidate is SPARK Ada, a subset of the Ada language. SPARK Ada has a formal semantics defined in Z, tool support from SPARK Examiner static analysis tool, and the strong type system of Ada.

31. Young, D., “RTCA/DO-254: No Hiding Place for Avionics Suppliers?” VMEbus Systems, February 2004.

Identified Problem: The integrity of a safety-critical system is rooted in understanding and managing risk. At the hardware side, the risk could involve component failure, hardware design error, underestimated margins, thermal stress, mechanical integrity, latent defect, or unpredictable behavior. These issues are especially critical, when more complexity is being incorporated into avionics systems in the form of processors, graphic devices, bridges, ASICs, FPGAs, and memory parts. In addition, the designer must make use of off-the-shelf components to construct a processing subsystem, memory, buses, and external interfaces that are only tenuously traceable back to top-level system functions.

Suggested Solution: The introduction of commercial off-the-shelf (COTS) VMEbus products with off-the-shelf firmware and real-time operating system developed to DO-178B guidelines, with the right quality levels and design assurances, would offer more cost-effective solutions than the architectural systems based on redundancy. Recognizing that COTS products could have a place in safety-critical avionics systems even where DO-254 is a requirement, the Avionics Process Management Committee has produced the EIA-933 Standard for Preparing a COTS Assembly Management Plan. This document recommends how to select and manage suppliers of avionics COTS products.

32. Hilderman, V. and Baghai, T., “Avionics Hardware Must Now Meet Same FAA Requirements as Airborne Software,” *COTS Journal*, September 2003.

Identified Problem: Until recently, only airborne software had to comply with rigid FAA design assurance and verification process steps, with certification based on DO-178B guidelines. However, avionics hardware was not required to meet such strict requirements, so functionality could be moved from software to hardware to avoid the rigors of DO-178B.

Suggested Solution: The paper gives a general introduction to DO-254 and lists three supporting processes important for certification: configuration management, process assurance, and certification liaison. It is advised to support configuration management by public open source tools: concurrent version system (CVS) and a bug tracking system. CVS is a revision control system that maintains a history of changes to the controlled project. It records who makes a change, the date, and reasons for the change. GNATS is a problem-reporting tool. Problem reports are submitted via email and are automatically logged into a database and forwarded to a responsible party. Regarding process assurance, the recommended strategy is to focus on ensuring correctness at the conceptual

design stage and then preserve the design integrity as one proceeds through detailed design and implementation. The certification basis depends upon the conceptual design, which is maintained under configuration management.

33. Thornton, R.K., “Review of Pending Guidance and Industry Findings on Commercial Off-The-Shelf (COTS) Electronics in Airborne Systems,” Report DOT/FAA/AR-01/41, FAA Office of Aviation Research, Washington, DC, August 2001.

Identified Problem: The use of complex electronic hardware components in airborne systems poses a challenge to meet safety requirements because, for complex components, complete verification is, at best, very difficult and, at worst, not achievable. Using COTS components in airborne systems raises a number of issues with respect to meeting airborne systems safety requirements and DO-254 objectives. In addition, commercial market trends are rapidly diverging from the needs of safety-critical airborne systems. The current move towards SoC designs, which may incorporate close to one million gates, has sparked the development of a new wave of EDA tools that will enable the trend towards more complex commercial microelectronics. While the new tools may provide some increased level of design assurance, qualification of these tools is an issue.

Suggested Solution: About a dozen key component attributes have been identified to meet the DO-254 objectives and guidance. One of them is the “role of COTS tools in design and verification.” The report identifies the following aspects of their role:

- Appropriate design models for formal methods
- Assessed tools for formal methods
- Assessed development tools
- Assessed verification tools
- Employ qualified verification tool
- Employ qualified design tools

The report identifies two essential ways of design verification: simulation and formal verification, and lists six references to access more detailed information. In conclusion, the report states that qualification of the tools to meet objectives for critical levels A and B in DO-254 was not evidenced in this investigation and may present a barrier to meeting the objectives, unless the development effort required to qualify the tools is undertaken or additional assurance can be gained by other means (pp. 89).

34. Lee, C., “IPT Guidance for Acquisition of Systems With Complex Programmable Hardware Using DO-254,” ERA Technology Ltd, June 2007.

Identified Problem: For UK military systems, the safety assurance of AEH has been specifically addressed by Def Stan 00-54 introduced in 1999. However, the standard was withdrawn in December 2004 and replaced by the system-level Def Stan 00-56 issue 3. It was thought that the less prescriptive approach to system safety assurance would facilitate the development and certification of novel systems with AEH. One unintended effect of

this approach to safety assurance has been the removal of detailed guidance for the specification and procurement of AEH. For a rapidly developing technology, guidance is required for most suppliers at some stage and its lack may actually discourage its exploitation due to the perception of increased project risk.

**Suggested Solution:** The report aims to guide the procurement and acceptance of military avionic systems based on the continuing technical advances that are being made in electronic system design, in general, and the capabilities of PLDs in particular. An interpretation is given of DO-254 in a view of military systems, quoting several common issues in DO-254 development and certification, such as:

- Inadequate level of detail in requirements
- Inadequate formal planning and following of plans
- Lack of independence in quality assurance and verification
- Inadequate and nonautomated traceability
- Lack of automated testing.

The issue of tool qualification is not addressed in this report.

35. Baghai, T. and Burgaud, L., “DO-254 Package: Process and Checklists Overview and Compliance With RTCA/DO-254 Document,” March 2004.

**Identified Problem:** DO-254 was issued several years ago. Although it established the standard for qualification of AEH, it remains vague in several aspects, and clarification is needed regarding those ambiguous issues. An interpretation of the standard needs to be given for actual examples from industrial practice.

**Suggested Solution:** A DO-254 Users Group has been established to help identify and resolve common problems. The DO-254 package includes the following five items designed to assist in the qualification process:

- The process documents help define, benchmark, and improve the industrial design, verification, validation, and quality assurance processes.
- The quality assurance checklists, for reviews and audits, ensure that each project is compliant with the defined industrial process.
- The tools
  - Reqtify™ for requirements management and traceability
  - VN-Check for checking compliance of HDL code with coding standards
  - VN-Cover for HDL code verification
  - VN-Optimize for test suite optimization to increase productivity

- The integration of the tools into the industrial process before their qualification (interfaces, report generation for a certification audit, training, tools assessment, etc.).
  - The DO-254 training by consulting partners.
36. Burgaud, L., “The DO-254 Users Group: A Proactive Initiative to Federate Industry Efforts,” Presentation at the FAA Software & AEH Conference, New Orleans, LA, July 2007.

Identified Problem: The DO-254 added new objectives and challenges to the hardware design processes. Requirements management became mandatory in hardware processes. The avionics and aerospace industries needed to establish partnerships and potentially share expertise and process improvement plans via structured collaboration.

Suggested Solution: A DO-254 Users Group has been established to help identify common problems and resolve them. The presentation outlines objectives, membership, roadmap, and some of the DO-254-oriented improvements in the design process. Xilinx, Altera<sup>®</sup>, TNI-Software, and Mentor Graphics present their individual slides. In particular, TLI-Software identifies their tools to support the process: Reqtify<sup>®</sup> (a requirements traceability and requirements-based engineering tool), RT-Builder<sup>™</sup> (a real-time architecture modeling and simulation tool), Eclipse<sup>®</sup>-based solutions and workbenches, and others.

37. Lundquist, P., “Certification of Actel Fusion According to RTCA DO-254,” Master Thesis, Report LiTH-ISY-EX-ET-07/0332-SE, Linköping University, Sweden, May 4, 2007.

Identified Problem: In recent years, the aviation industry moved toward using PLDs in airborne safety-critical systems. To be able to certify the close to fail-safe functionality of these programmable devices (e.g., FPGA) to the aviation authorities, the aviation industry uses DO-254 guidance for design assurance for AEH. At the same time the PLD industry is developing ever more complex embedded SoC solutions integrating more and more functionality on a single chip. This thesis looks at the problems that arise when trying to certify SoC solutions according to DO-254. Used as an example of an embedded FPGA, the Actel<sup>®</sup> Fusion<sup>™</sup> FPGA chip with integrated analog and digital functionality is tested according to the verification guidance.

Suggested Solution: Standard FPGAs, programmed using Verilog or VHDL languages, are used today in several real airborne safety-critical systems. For example, more than 700 Actel FPGAs are used in the Airbus A380 commercial airliner. That a certification procedure for a standard non-embedded FPGA-based, safety-critical system is possible has been shown in this thesis. The programmable logic industry will continue to design SoC solutions for the market, including soft processors, analog and digital amplifiers, communication interfaces, and filters. If these solutions could be used in the aviation

industry, it would mean using fewer systems that could do more, thereby, among other things, reducing system complexity and developing costs. The question of how these embedded chips could pass certification to be used in safety-critical systems remains unanswered.

38. Jacklin, S., et al., "Development of Advanced Verification and Validation Procedures and Tools for the Certification of Learning Systems in Aerospace Applications," *Proc. AIAA Infotech@Aerospace 2007 Conference and Exhibit*, Arlington, Virginia, September 26-29, 2005, Paper No. AIAA 2005-6912.

**Identified Problem:** Highly advanced adaptive control systems are needed to fulfill the present and future aerospace needs of the nation. Adaptive control technologies that incorporate learning algorithms have been proposed to enable automatic flight control and vehicle recovery, autonomous flight, and to maintain vehicle performance in the face of unknown, changing, or poorly defined operating environments. For civil aviation, adaptive control systems have been proposed that use learning to recover loss of vehicle control due to sudden aircraft damage or component failure. For robotic applications, the ability to learn gives adaptive control systems greater capability to adapt to changing mission requirements after deployment. Adaptive control systems have virtually unlimited applications for NASA space exploration applications, including mated flight vehicle coordination, docking, and control of autonomous robots, flyers, and satellites. Because most of these applications are in safety-critical areas, it is obvious that adaptive control systems with learning systems will never become part of the future unless it can be proven that this software is very safe and reliable. Rigorous methods for adaptive software verification and validation must be developed by NASA and others to ensure that control system software failures will not occur, to ensure the control system functions as required, to eliminate unintended functionality, and to demonstrate that certification requirements can be satisfied.

**Suggested Solution:** The FAA certification requirement to show that learning software programs meet their intended function do not negatively impact other systems or functions on the aircraft and are safe for operation, as pointed out in DO-178B, involves more than just running a set of test cases. The complete verification and validation of learning systems should not be viewed as running test cases and comparing expected results to actual results because such tests can never reveal the absence of errors. The verification and validation objectives must be satisfied by a combination of reviews, analyses, the development of test cases and procedures, and the subsequent execution of those test procedures. Simulation and methods to automate simulation remain very important tools because, at present, only they can really test and explore the most nagging problems of adaptive system verification, such as algorithm stability and convergent learning. Yet, the fact that testing can never reveal the absence of errors is a major shortcoming of this approach. Therefore, future progress toward certification requires that a number of new tools, such as the ones cited herein, be developed to allow the ultimate certification of adaptive control systems that use learning algorithms. In all

likelihood, a combination of analysis, tools, and simulation will be needed to address the full aspect of the certification problem for learning systems.

39. Crum, V., Homan, D., and Bortner, R., "Certification Challenges for Autonomous Flight Control Systems," *Proc. AIAA Guidance, Navigation, and Control Conference and Exhibit*, Providence, RI, August 16-19, 2004, Paper No. AIAA 2004-5257.

Identified Problem: As the U.S. Air Force works toward developing intelligent and autonomous weapon systems, a daunting task looms. How can one certify that a decision-making intelligent system is safe when the decisions are unpredictable? Trusting decisions made by autonomous control software will require completely new methods and processes to guarantee safety. The difficulty lies in determining how these intelligent systems will operate in a dynamic environment and with less human oversight. New paradigms will be needed to assure safety. Certification of flight control technologies is already the most rigorous testing embedded computer systems endure. Intelligent control adds a whole new dimension of issues. Adding intelligence can be divided into three challenges: building intelligence, instilling safety, and enabling affordability. All three are closely related. Cost and safety issues will influence how one designs and builds intelligence. Unmanned aerial vehicle (UAV) autonomous control is a revolutionary leap in technology. Such control replaces decision-making that required years of training for human operators. Neglecting autonomous control certification research today will dramatically increase tomorrow's cost of ownership for future users.

Suggested Solution: There are many technical challenges associated with certification of intelligent and autonomous control systems. Advanced UAV capabilities being developed today will challenge certification techniques far beyond their current capacities. New validation and verification (V&V) technologies are needed to enable timely and efficient certification of the intelligent and autonomous UAV control systems still in their infancy. V&V tools are needed to achieve the necessary degree of rigor that will ensure safety and mitigate risks associated with implementing autonomous control. A lack of research investment in certification technologies will have a significant impact on levels of autonomous control approaches that can be properly flight-certified and could lead to limiting capability for future autonomous systems. In addition, these advances in certification must also be repeatable to ensure that modifications to the control system cannot directly or regressively compromise airworthiness. The aerospace community has acknowledged that a consolidated research and development effort will be required to adequately address certification challenges and to share the investment burden to realize technological change in the certification process.

40. Baghai, T. and Burgaud, L., "Reqtify: Product Compliance With RTCA/DO-254 Document," May 2006.

Identified Problem: Reqtify is an effective solution for requirement traceability, impact analysis, and automated documentation generation. Reqtify supplies the following functionalities: requirement coverage analysis, upstream and downstream impact

analysis, requirement change, update and deletion tracking throughout the project life cycle, requirement attribute handling, filtering and display depending on these attributes, user-configurable documentation generation, and regression analysis. This technical note presents how Reqtify complies with the DO-254 objectives.

**Suggested Solution:** According to DO-254 classification, Reqtify is a verification tool, as it is a tool “that cannot introduce errors, but may fail to detect an error in the hardware item or hardware design.” Prior to using the tool, a tool assessment should be performed. The purpose of tool assessment and qualification is to ensure that the tool is capable of performing the particular verification activity to an acceptable level of confidence for which the tool will be used. It is only necessary to assess those functions of the tool used for a specific hardware life cycle activity, not the entire tool. The assessment activity focuses as much or more on the application of the tool as the tool itself. The verification tool only needs to be qualified if the function that it performs is not verified by another activity. The flow chart from DO-254 is applied and indicates the tool assessment considerations and activities and provides guidance for when tool qualification may be necessary.

41. Aldec<sup>™</sup>, Inc., “DO-254 Hardware Verification: Prototyping With Vectors Mode,” June 26, 2007.

**Identified Problem:** A sample design includes a counter, with the following features: one clock domain, asynchronous reset, clock-enable port, counting direction port (up/down), synchronous initial value reload ability, and 64-bits output data. The system contains two boards connected through daughter board connectors. The main board is an Aldec HES<sup>™</sup> board (HES3X3000EX), which is connected to the PCI bus. This board generates stimuli for the device under test (DUT) and collects results from the DUT. The second board is a user daughter board with DUT.

**Suggested Solution:** The verification process contains three independent stages: simulation, verification, and comparison. The simulation stage is a typical HDL-level simulation in Active-HDL simulator. During simulation, stimuli and results are captured to a waveform (ASDB format) on a specified edge of a user clock. The clock line of the DUT is not stored in the waveform file. It is generated on the HES main board during verification to assure constant frequency. For hardware verification purposes, the PrototypeVerificationTool program is used to send test vectors to the DUT and retrieve response data from the DUT. During the verification process, the application continuously performs two tasks: writing stimuli to SinFIFO and reading results from RoutFIFO. The results from RoutFIFO are written to a raw binary file. At the end of verification, the binary results are transformed to an ASDB waveform file. At the comparison stage, the waveform captured during simulation is compared with the waveform obtained from the hardware verification. If there is no differences, it means that verification has finished successfully. The Aldec waveform viewer, the Wvcore, can be used for waveform comparison.

42. Leroy, J.-E. and Bezamat, J., “Experience at Barco-Silex in FPGA Design With DAL C (DO254),” Barco-Silex S.A., Peynier, France, Internal Paper, 2007.

Identified Problem: Designing FPGAs for AEH should be compliant with DO-254 rules. This design assurance implies changes in traditional design rules. Improvements of the methodology are necessary for circuit development. From communication within the project toward test bench methodologies, many things will be handled differently by the design team, thus implying the increase of documentation and, therefore, the final cost.

Suggested Solution: The paper gives an overview of the way Barco-Silex has handled the DO-254 constraints for designing several FPGA circuits. It addresses the development cost impact on FPGA design throughout the verification level and the amount of data delivered in this process. The golden rule to provide hardware design assurance for any design entity is to split the three fundamental design rules, which are: specification, conception, and validation. These must be assumed by different people to avoid error propagation from the beginning until validation. In many cases, validation results may need to be reviewed independently to confirm that proper procedures were followed and the results confirm the requirements have been met.

43. Pampagnin, P. and Menis, J.F., “DO254-ED80 for High Performance and High Reliable Electronic Components,” Barco-Silex S.A., Peynier, France, Internal Paper, 2007.

Identified Problem: Today, avionics manufacturers follow the rules given by nonairborne markets (telecom, personal computer, multimedia, and home electronics). These strong-market leaders are driving the entire electronics domain, including components procurement, computer-aided design tool usage, and methodology implementation. These leading markets have very short life cycles compared to aircraft (for instance, life cycle for a memory is around 18 months, but the life cycle of an aircraft is 30-40 years). Taking into account the technology changes, avionics designers have to also cope with such trends as:

- New and novel technology issues
- Merging formerly separate and independent functions on the same hardware
- Multifunction components
- Displaying critical and noncritical functional paths in the same systems and components
- Replacing mechanical with electronic parts (for example, relays and switches)
- Using AEH in roles traditionally targeted at software

- Configuring control of complex and highly integrated systems. This imposes a very complicated environment to apply DO-254/ED-80 procedures to the design process.

**Suggested Solution:** Even if implementing DO-254/ED-80 has a negligible cost, this can be considered an investment. It obliges the supplier to analyze in detail its processes, methodologies, and tools and to apply structured development processes, with a rigorous quality assurance. It also allows the supplier to adapt its set of internal processes to the DAL targeted to optimize efforts. The resulting products have a better quality and the development cycles are optimized. Verification is focused on design errors, and effort and resources are better distributed. The subcontractor is thus obliged to respect a structured development processes. The initial cost has to be compared with the level of quality for the subcontractor. Applying DO-254/ED-80 ensures that the subcontractor will provide a high level of quality, selected documentation, and the ability to reuse the design, if necessary.

44. Dellacherie, S., Burgaud, L., and di Crescenzo, P., “imPROVE—HDL: A DO-254 Formal Property Checker Used for Design and Verification of Avionics Protocol Controllers,” *Proc. DACS’03, 22nd Digital Avionics Systems Conference*, Indianapolis, Indiana, October 12-16, 2003, Vol. 1, pp. 1.A.1-1.1-8.

**Identified Problem:** An airplane today contains a large network linking embedded controllers to sensors/actuators and communications equipment onboard. Efforts made in recent years to simplify network wirings have resulted in significant reductions in the aircraft weight and labor required to run wiring harnesses. This has often come at the cost of more complex data bus architecture (bidirectional protocol instead of unidirectional protocol). DO-254 considers the use of formal methods and requirements traceability when developing hardware to support safety-critical (level A or B) functions. This paper looks at a static formal approach that may be used, in combination with requirements traceability features, to apply formal methods in the design and verification of hardware controllers to support protocols such as ARINC 429, ARINC 629, MIL-STD-1553B, for example.

**Suggested Solution:** This paper describes the application of a formal tool, imPROVE-HDL, in the design and verification of AEH developed in a DO-254 context. imPROVE-HDL is a formal property checker that complements simulation in performing exhaustive debugging of VHDL/Verilog Register-Transfer-Level hardware models of complex avionics protocol controllers without creating test benches. The Reqtify tool is used to track the requirements throughout the verification process and to produce coverage reports. Using imPROVE-HDL coupled with Reqtify, avionics hardware designers are assured that their bus controllers meet the most stringent safety guidelines outlined in DO-254.

45. Lange, M., “Automating Clock-Domain Crossing Verification for DO-254 (and Other Safety-Critical) Designs,” White Paper, Mentor Graphics Corporation, December 2007.

Identified Problem: “Metastability” is the term used to describe what happens in digital circuits when the clock and data inputs of a flip-flop change values at approximately the same time. This leads to the flip-flop output oscillating and not settling to a value within the appropriate delay window. In this case, the output of the DC flip-flop is said to have gone “metastable.” This situation happens in every design containing multiple asynchronous clocks, which occurs any time two or more discrete systems communicate. Metastability is a serious problem in safety-critical designs in that it frequently causes chips to exhibit intermittent failures. To understand CDC in the context of DO-254, the purpose of DO-254—design assurance—should be considered.

Suggested Solution: A comprehensive CDC verification solution, such as 0-In CDC, must do three distinct things:

- Perform a structural analysis. This is most effectively done on the register transfer language code to identify and analyze all signals crossing clock domains, and determine if their synchronization schemes are present and correct.
- Verify transfer protocols. This ensures that the synchronization schemes are used correctly by monitoring and verifying that protocols are being followed during simulation.
- Globally check for reconvergence. This is most effectively done by injecting the effects of potential metastability into the simulation environment and determining how the design will react.

0-In CDC provides added assurance that the design will function correctly within the intended system (this is the intent of DO-254). However, unless a specific requirement has been identified by the customer that states one must verify the CDCs, 0-In CDC can be run without it becoming part of the DO-254 review process. On the other hand, if a specific requirement from the customer (or the designated engineer representative) states the CDCs must be verified to identify and eliminate instances of metastability, then a method of tool assessment must be chosen. The simplest one is Independent Output Assessment.

## APPENDIX E—HARDWARE CASE STUDY EXPERIMENTS

The case studies are designed to be a subset of tests that would be run during an airborne electronic hardware (AEH) tool qualification. The case studies will test the tool from design entry through synthesis to the actual hardware implementation. The case studies are constructed as small, focused experiments to analyze tool performance under worst-case conditions. These conditions enable the bounds of the tools' capability to be assessed under challenging conditions. Using small, focused experiments allows the case studies to be fully verified before the design is committed to hardware. The following case studies were selected:

- **Timing Constraints Analysis:** How closely does system timing match the timing constraints? How much margin (if any) exists to the constraint?
- **Wide Data Buses:** Large data buses can switch a large number of pins at the same time. Can large data busses cause any issues?
- **Undefined Input/Output (I/O) States:** Field-programmable gate arrays (FPGA) often contain more pins than are used in a design. What is happening to the unused pins?

A case study investigating radiation effects was considered, but there were no facilities available to address radiation effects. It was also decided that, although it was feasible to implement test cases for the problems identified in sections 8.5 through 8.7, physically implementing these case studies would offer no additional insights. Therefore, these case studies were also rejected.

Appendix E begins by discussing the FPGA architecture and the electrical hardware used to implement the case studies. It then examines each case study individually and draws conclusions from the test cases analyzed.

### E.1. ARCHITECTURE BACKGROUND FOR FPGA.

A typical FPGA I/O contains a large amount of circuitry. This circuitry is designed to allow a single pin to be either an input or an output for numerous I/O signaling standards. Outputs can have totem pole, open drain, or open source configurations as well as resistive pull-ups and pull-downs. Outputs can be operated on either one clock edge or both clock edges. Inputs can have resistive terminations and predefined or user-defined signaling thresholds. Both inputs and outputs can be operated on single-ended and differential signals.

The circuitry for a standard I/O is shown in figure E-1. For any given signaling configuration, only a small portion of the hardware is actually active. The remaining hardware is deactivated transparently to the user. The user may be unaware that the additional hardware exists and operate the device in a fashion where the unused hardware may impact device operation.

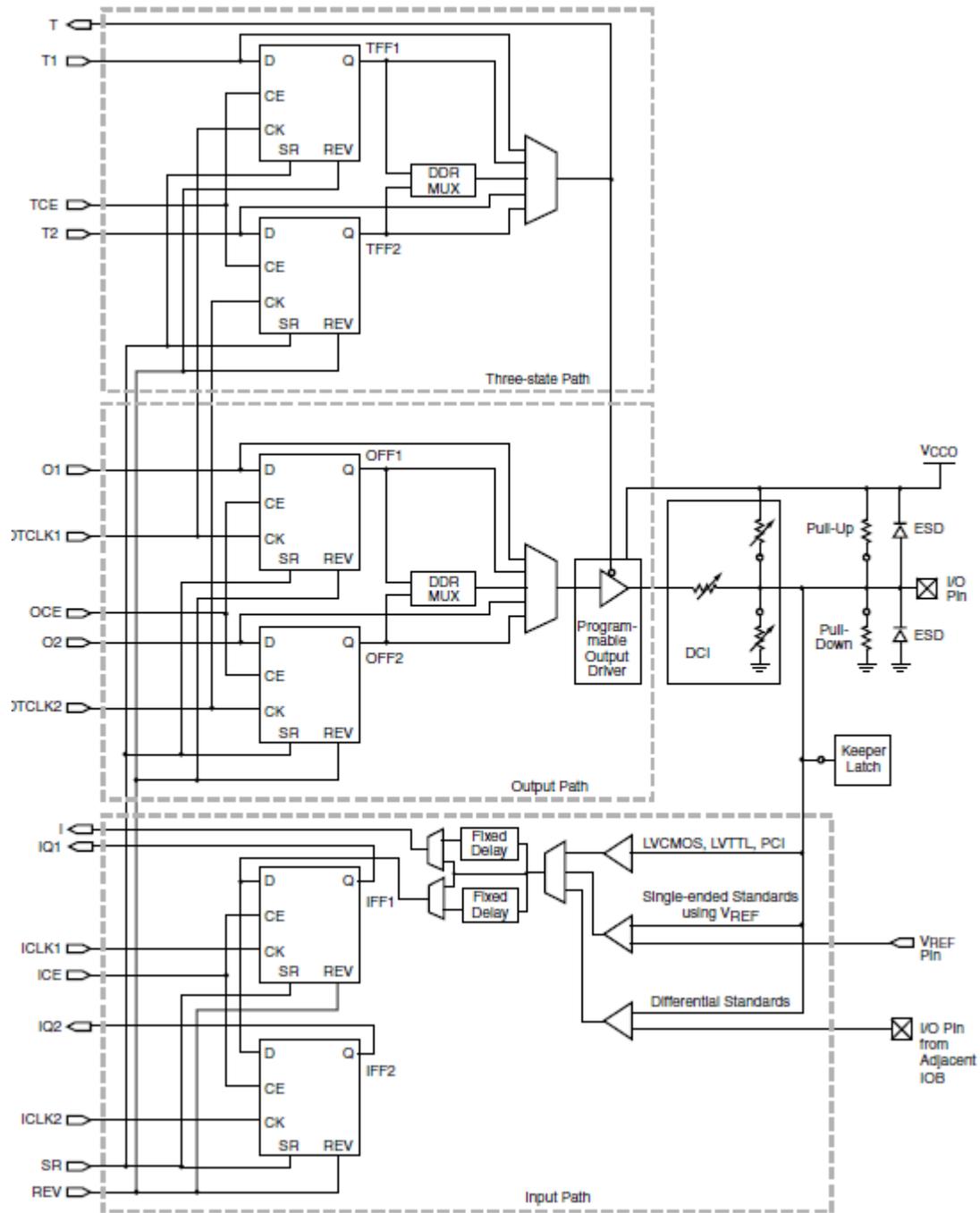


Figure E-1. The FPGA I/O Architecture for Standard I/O Configurations [E-1]

The Xilinx<sup>®</sup> Spartan<sup>™</sup>-3E FPGA family [E-1] is a low-cost family offering mid-level performance and capability. As shown in table E-1, a standard FPGA contains hundreds of I/O pins. To allow the FPGA to operate using multiple signaling standards, the I/Os are grouped into banks that share a common power supply voltage. The number of banks and I/Os per bank vary with the FPGA and the FPGA package.

Table E-1. The Gate Count and Number of User I/Os for the Xilinx Spartan-3E FPGA Family [E-2]

| Device    | System Gates | Maximum User I/O |
|-----------|--------------|------------------|
| XC3S100E  | 100K         | 108              |
| XC3S250E  | 250K         | 172              |
| XC3S500E  | 500K         | 232              |
| XC3S1200E | 1200K        | 304              |
| XC3S1600E | 1600K        | 376              |

One of the safety issues targeted for testing was the possibility of simultaneous switching noise (SSN) introducing errors. Therefore, it was necessary to select a hardware platform representative of current FPGA technology that could run all of the desired tests and also have enough outputs to be capable of generating SSN. FPGA manufacturers provide guidance on how many outputs per bank can be switched simultaneously. The I/O standard that is being used is a significant factor in determining the SSN limits, with the worst-case standard being low voltage transistor-transistor logic (LVTTL) I/Os. Figure E-2 shows the recommended maximum number of single-ended outputs for the LVTTL standard. The fewest number of I/Os per VDD/ground (GND) pair occur when the output is configured for a fast slew rate and a high drive strength. Figure E-3 shows that number of VDD/GND pairs per I/O bank for several devices in the Spartan 3-E family. The fewest number of VDD/GND pairs occurs for packages with the fewest number of total pins.

| Signal Standard (IOSTANDARD)  |      | Package Type |           |           |           |                                     |    |
|-------------------------------|------|--------------|-----------|-----------|-----------|-------------------------------------|----|
|                               |      | VO<br>100    | TO<br>144 | PO<br>208 | CP<br>132 | FT256,<br>FG320,<br>FG400,<br>FG484 |    |
| <b>Single-Ended Standards</b> |      |              |           |           |           |                                     |    |
| LVTTL                         | Slow | 2            | 34        | 20        | 19        | 52                                  | 60 |
|                               |      | 4            | 17        | 10        | 10        | 26                                  | 41 |
|                               |      | 6            | 17        | 10        | 7         | 26                                  | 29 |
|                               |      | 8            | 8         | 6         | 6         | 13                                  | 22 |
|                               |      | 12           | 8         | 6         | 5         | 13                                  | 13 |
|                               |      | 16           | 5         | 5         | 5         | 6                                   | 11 |
|                               | Fast | 2            | 17        | 17        | 17        | 26                                  | 34 |
|                               |      | 4            | 9         | 9         | 9         | 13                                  | 20 |
|                               |      | 6            | 7         | 7         | 7         | 13                                  | 15 |
|                               |      | 8            | 6         | 6         | 6         | 6                                   | 12 |
|                               |      | 12           | 5         | 5         | 5         | 6                                   | 10 |
|                               |      | 16           | 5         | 5         | 5         | 5                                   | 9  |

Figure E-2. Recommended Maximum Number of Single-Ended Outputs per VDD/GND Pair [E-3]

| Device    | Package Style (including Pb-free) |       |       |       |       |       |       |       |
|-----------|-----------------------------------|-------|-------|-------|-------|-------|-------|-------|
|           | VQ100                             | CP132 | TQ144 | PQ208 | FT256 | FG320 | FG400 | FG484 |
| XC3S100E  | 2                                 | 2     | 2     | -     | -     | -     | -     | -     |
| XC3S250E  | 2                                 | 2     | 2     | 3     | 4     | -     | -     | -     |
| XC3S500E  | 2                                 | 2     | -     | 3     | 4     | 5     | -     | -     |
| XC3S1200E | -                                 | -     | -     | -     | 4     | 5     | 6     | -     |
| XC3S1600E | -                                 | -     | -     | -     | -     | 5     | 6     | 7     |

Figure E-3. The Number of VDD/GND Pairs per I/O Bank for the Xilinx Spartan-3E FPGA Family [E-3]

## E.2 THE COMPLIMENTARY METAL-OXIDE-SEMICONDUCTORS LVTTL STANDARD.

For the 3.3 V LVTTL standard, the maximum voltage that is guaranteed to be interpreted as a 0 is 0.8 V. (The maximum voltage for a low input is called VIL.) The minimum voltage that can be interpreted as a 1 is 2.0 V. (The minimum voltage for a high input is called VIH.) Input circuits are designed to have a nominal trip voltage of about 1.4 V. This means that all voltages above 1.4 V are interpreted as a 1 and all voltages below 1.4 V are interpreted as a 0.

In complimentary metal-oxide-semiconductors (CMOS) processes, the input trip voltage is a function of the applied power supply voltage. For instance, an input circuit with a trip voltage of 1.4 V and a 3.3 V power supply would have a trip voltage of  $0.42 * VDD$ . If this device is operated at 3.0 V, the trip voltage would decrease to 1.27 V. SSN can induce voltages on the power and ground supplies, which can cause the trip voltages to change and possibly lead to an erroneous circuit operation.

In a CMOS process, numerous parasitic diodes exist. In normal operation, the parasitic diodes are reverse biased. This results in a substantial internal capacitance between the internal VDD and the internal ground. Parasitic inductances in the power and ground supply grids effectively decouple the on-die VDD and ground voltages from external supplies. The resulting on-die power grid maintains the difference between the internal VDD and internal ground, even if the internal ground level varies. If the difference between VDD and internal ground is 3.3 V and SSN causes the internal ground to rise up by 0.6 V, then the internal VDD will also rise by 0.6 V.

## E.3 EXPERIMENTAL HARDWARE PLATFORM.

The original board selected for this project was the Spartan-3E starter board, which offered connections to 38 I/Os on a single bank. For the FG320 package used on the Spartan-3E starter board, the Xilinx SSN design tool guidance allows up to 45 SSOs on a bank [E-1]. Therefore, this board would not allow the capabilities of the internal FPGA power networks to be stressed. It should be noted that the experiment is limited by the outputs the evaluation board makes available to the user. A custom-designed board for the FPGA could easily exceed the SSN tool design guidance. Many different FPGA evaluation boards were examined, and numerous trade-offs ensued. The boards with the best access to large numbers of outputs in a single bank had devices in large pin count packages that reduced the susceptibility to SSN effects. Boards with smaller pin count packages had restricted access to the I/O pins. None of the evaluation boards

met the needs of this research. The constraints placed by the evaluation board limitations would not be a concern for a real design, since a custom board could be designed. The research schedule did not allow for the time to design a custom board for the FPGA evaluation. After much deliberation, it was decided to use the Spartan-3E starter board and design a custom load board that would provide the ability to connect a variety of loads to the output pins, give access to the necessary test signal levels, and provide monitors for the signal integrity.

The Spartan-3E starter board was selected as being similar enough to the originally selected test board and did not require major revisions to the test cases, providing the capability of stressing the SSN guidelines. The Spartan-3E starter board provides connections to 20 I/Os on a single bank. The Xilinx guidance for the maximum number of I/Os that can be simultaneously switched is 21 I/Os, configured as 24 mA fast slew LVTTTL outputs per bank [E-3]. This board can never violate the Xilinx-recommended loading; therefore, if any failures occur, they will be unexpected.

The load board is intended to connect to a variety of evaluation boards via a standard board expansion connector. For the Spartan-3E starter board, this connector provides access to the FPGA bank 0 I/O, as well as some of the bank 1 I/Os.

As shown in figure E-4, the FPGA I/Os (pins D5, D6, and E7) are connected to a termination voltage ( $V_{term1}$ ). In normal operation,  $V_{term1}$  is connected via a jumper on header 3 to either VDD, ground (GND), or an external reference driving  $V_{term1}$ . The net result is that the user can connect a capacitive load to VDD, GND, or some other termination voltage reference. In addition, the board can be assembled with resistors instead of capacitors to provide resistive loads to VDD, GND, or some other termination voltage.

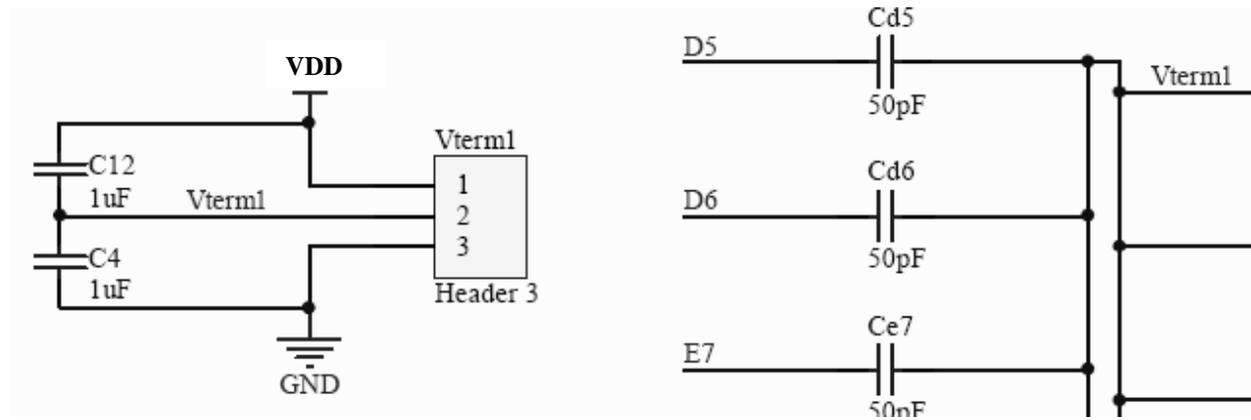


Figure E-4. The Custom Load Board Configuration

The termination voltage setting is applied to a group of I/Os, as described below:

- Connector A2 Vterm groupings
  - Vterm1: pins D5, D6, E7, D7, D8, D10, B4, E6, C5, C6, C8, C9, A3, and A4
  - Vterm2: pins B5, B6, A7, A8, B10, B11, A12, A13, A5, B7, B8, A9, A10, B12, B13, and B14
  - Vterm3: All connector B1 I/Os

### E.3.1 SIGNAL INTEGRITY MONITORING.

Signal integrity monitoring is necessary to assure that the signals are correct when SSN effects are not present; in addition, the monitors allow the effects of SSN to be observed. SSN monitor connections, as shown in figure E-5, are provided on I/O pins C6, A8, and B12. These monitors are implemented as 950- $\Omega$  series resistors to subminiature version A (SMA) connectors. The intended measurement equipment is a 50- $\Omega$  oscilloscope. The 950- $\Omega$  resistor plus 50- $\Omega$  scope provides a high impedance probe connection to the I/O. The oscilloscope should be set to a probe divide-ratio of x20.

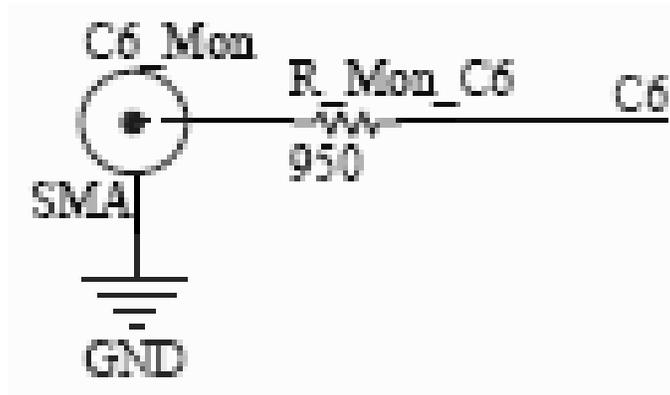


Figure E-5. Signal Integrity Monitor Schematic

### E.3.2 EXTERNAL DIRECT CURRENT INPUT VOLTAGE.

Input pin C7 is intended to be used as a direct current (DC) input to the FPGA (figure E-6). A Bayonet Neill Concelmun (BNC) connector is provided for connection to a laboratory power supply. Additionally, a header is provided to allow for connecting input C7 to either VDD or GND with a jumper.

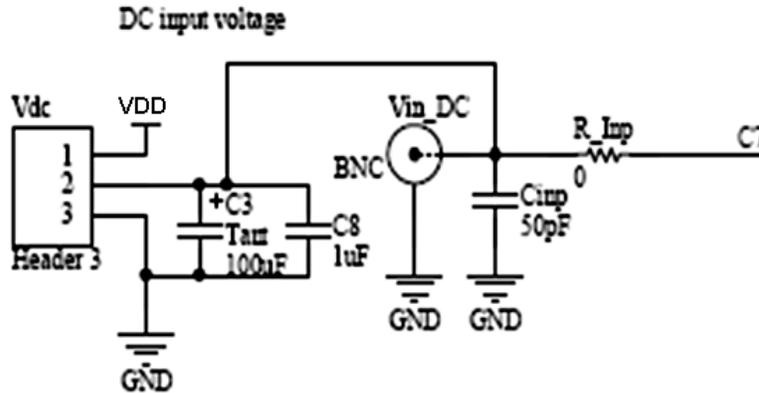


Figure E-6. External DC Voltage Input

### E.3.3 EXTERNAL CLOCK INPUT.

An SMA connector is provided for connectivity to I/O A8 (figure E-7). This is intended to be used as a clock input signal for timing constraint experiments. The FPGA pin is a global clock (GCLK) input and is specially designed to be used as a clock input.

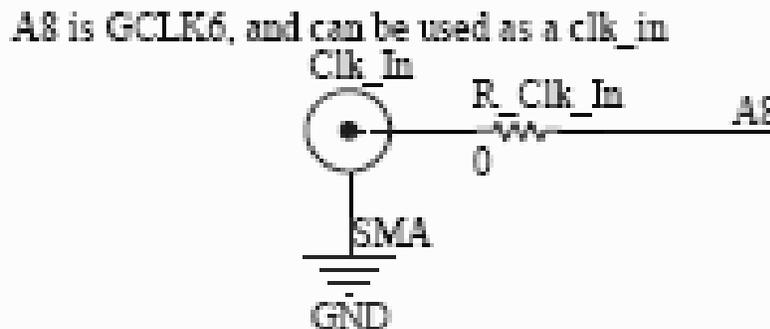


Figure E-7. External Clock Input

The custom-designed load board is shown in figures E-8 and E-9. Figure E-8 shows a close-up of the load board, and figure E-9 shows the load board attached to an FPGA evaluation board.

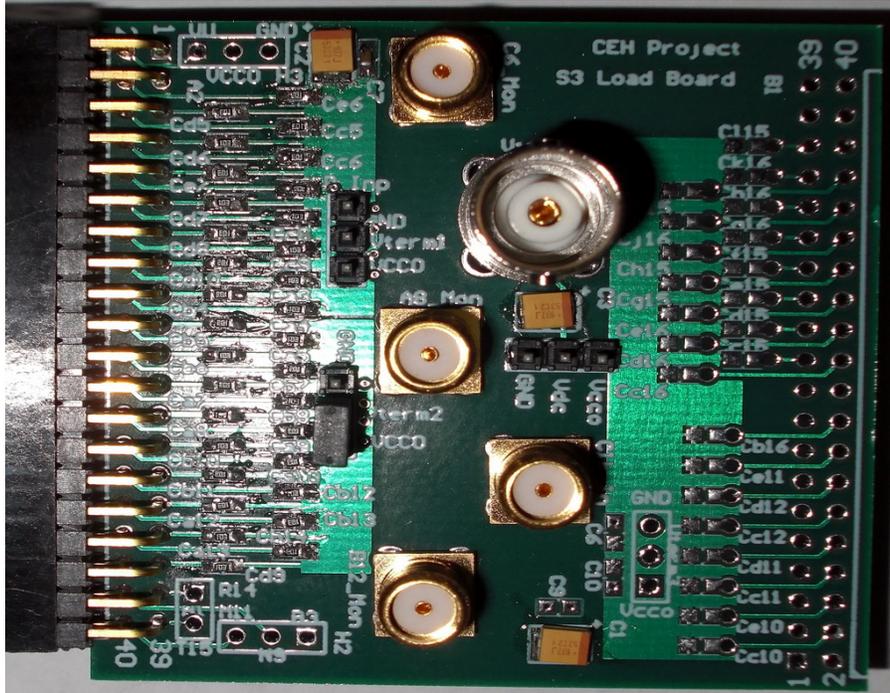


Figure E-8. A Close-Up of the Custom Load Board Showing the Connector Pins, Resistive Loads, and Monitoring Ports

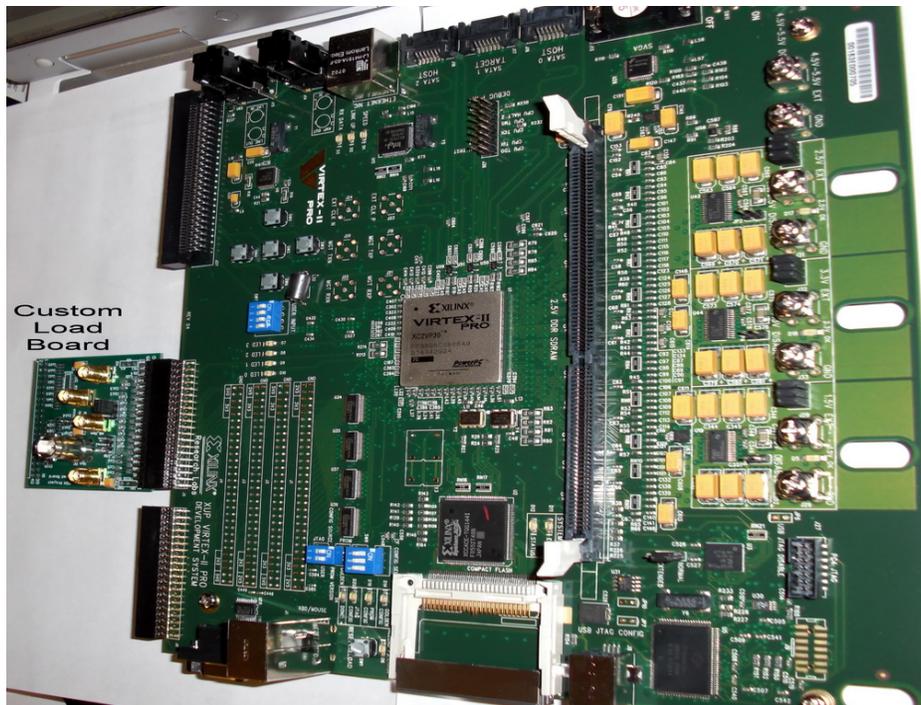


Figure E-9. An FPGA Evaluation Board Showing the Custom Load Board Connected to the Expansion Connector

#### E.4. EXPERIMENT DESCRIPTIONS.

The case studies were developed based on the expressed concerns of the scientific and industrial communities with reference to AEH tools, such as those used for FPGA development. Largely, these concerns were relevant to the development of safety-critical and real-time systems. These case studies were geared toward qualifying the AEH tools. In an attempt to qualify the tools, the tools will be used with worst-case scenarios, along with least-likely uses, to test the bounds of the tools' capability. The concept is that the black box design in the AEH tool will have a one-to-one mapping trace to the black box operation that is finally implemented. This, however, will be done in a design-independent fashion. To facilitate design independence, case studies were developed as small, focused experiments used to discover specific attributes of a tool. This method was preferred to large, elaborate designs because broken links in tracing from design to implementation were often caused by a flaw in the design. The following test case studies were identified:

- **Timing Constraints:** A determination of timing based on synthesis redesign used to establish the safety margin with respect to the timings reported by the tool.
- **Wide Data Buses:** A systematic way of determining if place-and-route functions are effective and drawing current in part of the circuit does not adversely affect other parts.
- **Undefined I/O States:** A test to determine how the tool uses the I/O pins not defined by the user.

In addition to the above, there are two other safety issues discussed. (1) A simulation error is the situation in which behavioral simulation results differ from actual implemented circuit behavior. Determination of accuracy and reliability of the simulation component of the tool is critical. However, good designers know the simulation is only one step and can be trusted only when the actual hardware testing confirms the simulation results. (2) Detection of faulty electronic hardware is the capability of the tool to notify the designer if the selected programmable logic device will not properly implement the synthesized circuit.

Thorough literature research, surveys, and industry interviews of uncertainties and faults regarding the usage and/or operation of the tools have been compiled and analyzed. The case studies are focused on verifying the validity of these findings. The scope of these findings includes user interaction with the tool, such as when a user tries to implement something physically impossible, if the tool notifies the user, alters the design to make it possible, or attempts to implement the design. This leads to other questions in the scope of the case studies: Does the tool have awareness of the hardware physical limits, or is this the responsibility of the user? For example, will the tool try to implement a component on a faulty piece of hardware? Will the tool exceed the minimum transition time of the gate timing or account for a safety margin? These are just a few of the many identified concerns, each of which can be traced to safety constraints or timing constraints.

The majority of the tools come in a package that contains everything from coding or formal requirements and design, to redesigning through synthesis, to testing the final implementation. The tools appear to be self-contained, in the sense that they do their own verifications and testing, including self-validation of formal requirements and design. Therefore, if the tool does all the design/redesign and it verifies its own design, it will not be known if it is truly correct and it is unclear if it has been independently verified as required by DO-178B and DO-254.

Also, can software or hardware itself verify through testing something that the software or hardware is not physically capable of actually doing? How much human intervention is needed to use these tools and what qualifications does the user need? In addition, the tools have no knowledge of the actual physical environment that the FPGA will be operating in and assume that everything is ideal. Is this a reasonable assumption for safety-critical systems? These are some of the questions that will be answered through an analysis of the case study results. Three of the previously identified case studies are described below.

### E.5 TEST CASE 1—SSN.

To test SSN performance, the FPGA will be configured with the maximum number of simultaneously switching outputs. It is expected that changing VDD and GND currents drawn by the outputs will cause the internal VDD and GND voltages to vary. To sense this variation in the internal power rail voltages, this test case will configure one of the I/Os in the same I/O bank to be an input. The threshold voltage of this input is a function of the power rail voltages. This input will be a constant voltage defined to be the maximum voltage allowed for a low input or the minimum voltage allowed for a high input. This input is then connected to an output on a lightly used I/O bank, as shown in figure E-10. If the SSN causes the input to misread its input signal, then the output on the lightly used bank will switch. This output will be monitored during the test. Any switching or unexpected signal transitions (glitching) of this signal indicates that SSN has produced a logical error.

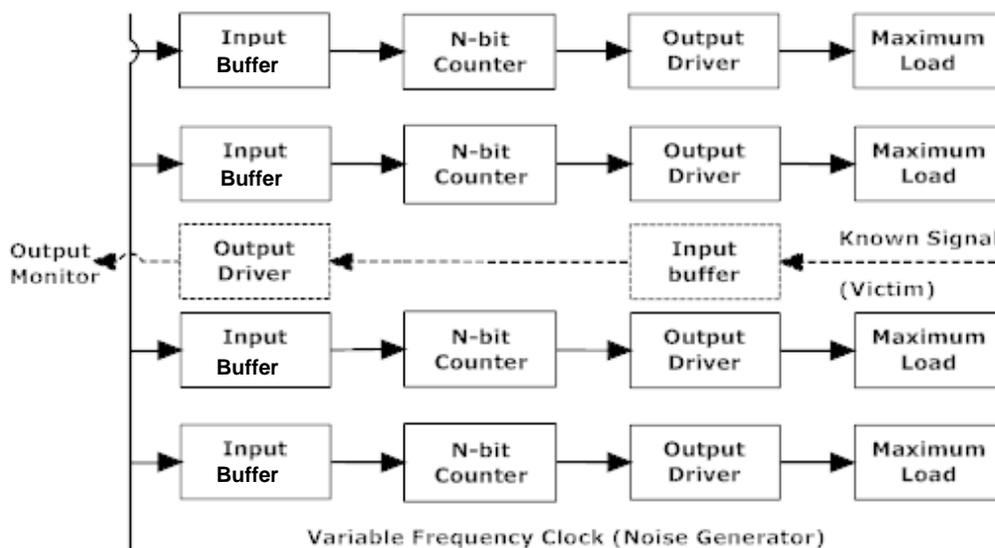


Figure E-10. Power Integrity and I/O Analysis Test Case

The SSN behavior will be examined with several different load configurations, because LVTTL loads can be modeled in several ways. First, the loads are rated at a load current of 24 mA. Resistive loads will be used to draw this much current from the I/O. For this case, a 130- $\Omega$  load will be used. Since it is not known whether switching from 0 to 1 or 1 to 0 is the worst case, the board will have the capability to terminate the resistor loads to either VDD or GND. This will allow both transitions to be easily investigated. The standard LVTTL switching load from an LVTTL part data sheet is a 50-pF capacitive load to GND. The effects of this capacitive load on SSN will also be measured.

The 19 outputs will switch slowly enough to guarantee that any transients produced by the switching will have decayed prior to the next switching event. The input will be connected to an externally referenced voltage. During each test, the input voltage is held constant. Since the input voltage is constant, the signaling level the input circuit observes should also be constant. If the input buffer observes any changes in the logic level, then an undesirable operation has occurred. The changes that are observed in the measured data are usually a rapid switching between logic levels. This response is called glitching, which is always undesirable since it can generate signals that do not meet the timing requirements of other circuitry in the device. From a safety-critical point of view, any glitching at all is a failure of the system. If the signal glitches when the input is above the  $V_{IH}$  level or below the  $V_{IL}$  level, then a violation of the LVTTL standard has occurred.

#### E.5.1 TEST CASE 1—IMPLEMENTATION DETAILS.

Project AEH\_Case1a was used to examine potential simultaneous switching output limitations. The project consisted of 19 Bank 0 I/Os (signal toggle\_out) on the FPGA simultaneously switching at a frequency of approximately 48 Hz. To monitor the FPGA for possible noise problems, the remaining Bank 0 I/O was configured as an input, dc\_in. Input dc\_in will be driven at the LVTTL specification limits of  $V_{IL} = 0.8$  V for a logic 0 and  $V_{IH} = 2.0$  V for a logic 1.

The project also contained logic to perform a latching function in the event of a high or low signal being detected on dc\_in. Signal latch\_low\_transition will turn on a light-emitting diode (LED) if a logic low is detected on dc\_in. Signal latch\_high\_transition will turn on an LED if a logic high is detected on dc\_in.

The state of dc\_in was echoed to an output signal, test\_out, which used an output on Bank 1 of the FPGA. The value of signal test\_out should always be constant, as signal dc\_in does not change. However, SSN could cause the logic level of dc\_in to be misread. This type of error will be recognized as a logical transition on the test\_out signal. The SSN experiments will be performed first with I/O loads of 130  $\Omega$ , then with loads of 50 pF. The load board must be populated as required by the individual experiments. Additionally, to examine the difference between low-to-high and high-to-low transitions with respect to SSN, the experiments may be performed with  $V_{term} = GND$  and  $V_{term} = VDD$  by changing the jumper settings of Vterm1 and Vterm2.

### E.5.2 TEST CASE 1—SSN RESULTS.

SSN experiments were performed with the following load conditions on signals toggle\_out:

- Resistive load  $R_L = 130 \Omega$ ,  $V_{term} = 0 \text{ V}$
- Resistive load  $R_L = 130 \Omega$ ,  $V_{term} = \text{VDD} (3.3 \text{ V})$
- Capacitive load  $CL = 47 \text{ pF}$ ,  $V_{term} = 0 \text{ V}$
- Capacitive load  $CL = 47 \text{ pF}$ ,  $V_{term} = \text{VDD} (3.3 \text{ V})$

For each of these conditions, the voltage on test input dc\_in was stepped from 0 to 3.3 V in 0.1 V steps; the logic level of test\_out was monitored using the Xilinx Chipscope™ Integrated Logic Analyzer (ILA) logic core. Using the ILA, test\_out was monitored to look for logical errors due to SSN. Screen captures from the ILA are shown in this report. The following three signals were monitored with ILA, and are shown in the ILA screen captures in figures E-11 through E-13.

- slw\_clk—A transition on this signal indicates a transition on the toggle\_out load signals.
- test\_out—DC output. A transition on this signal indicates SSN.
- counter<0>—Shown for reference, this is the LSB of the counter used to generate slw\_clk. This signal transitions on each rising edge of the system clock, and the resistive load  $R_L = 130 \Omega$ .

The performance of the output during glitching conditions was examined more closely. Figure E-11 shows the output glitching once for the case where  $V_{in\_dc} = 1.1 \text{ V}$ . Figure E-12 shows the output glitching many times at  $V_{in\_dc} = 1.4 \text{ V}$ .

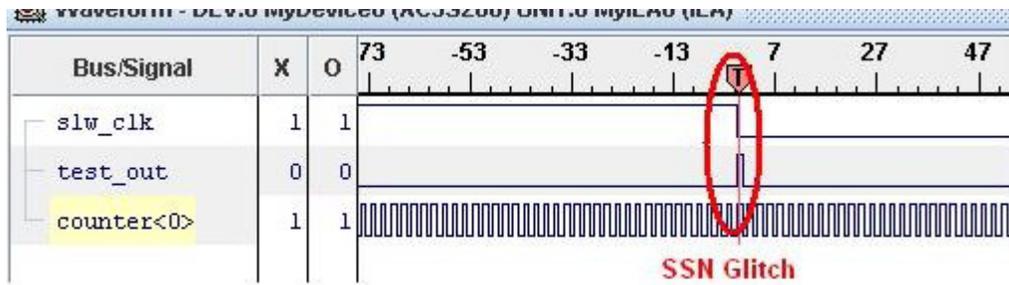


Figure E-11. Capacitive Loads With  $V_{term} = \text{GND}$  and  $V_{in\_dc} = 1.1 \text{ V}$  (a single glitch due to SSN was observed.)

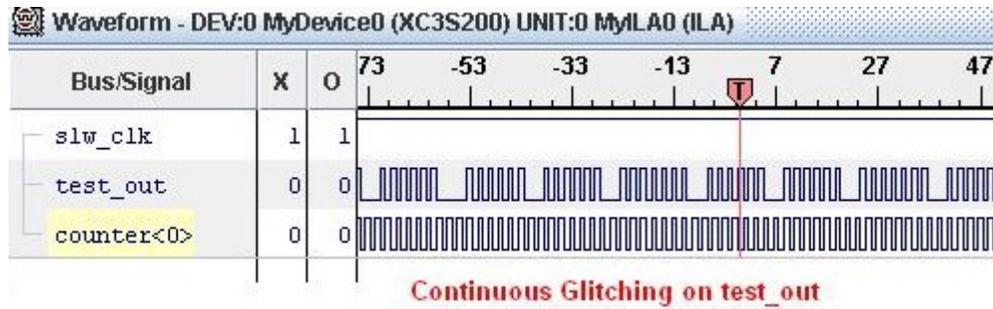


Figure E-12. Capacitive Loads With  $V_{term} = GND$  and  $V_{in\_dc} = 1.4\text{ V}$  (Many glitches due to SSN were observed.)

Glitching was also observed for  $V_{term}=VDD$ . Figure 30 shows a glitch when  $V_{in\_dc} = 2.3\text{ V}$ .

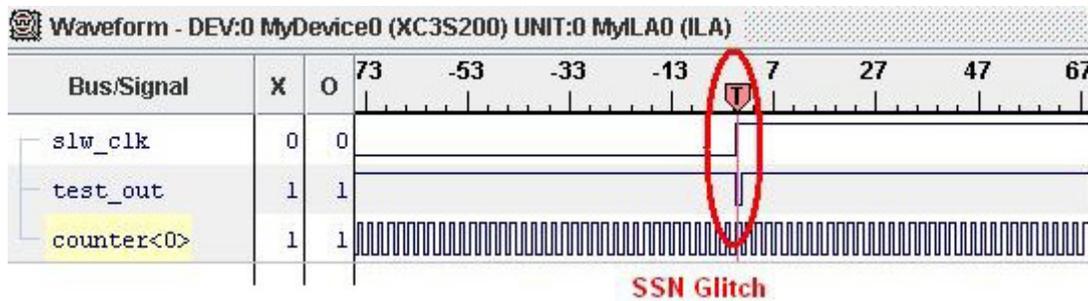


Figure E-13. Capacitive Loads With  $V_{term} = VDD$  and  $V_{in\_dc} = 2.3\text{ V}$  (A single glitch due to SSN was observed)

When testing with a resistive load, with  $V_{term} = 0\text{ V}$  and  $V_{term} = VDD$ , no SSN specification violations were observed. Table E-2 indicates the logic level reported on test\_out across the various input voltage test levels. The decision boundary between a logical 1 and a logical 0 was between 1.4 and 1.5 V, which is well within specifications for both  $V_{IL}$  and  $V_{IH}$ .

Table E-2. Logical Output Voltages as a Function of Input Voltage of the Nonswitching Input for Resistive Loads

| V <sub>in_dc</sub><br>(Volts) | <i>test_out</i><br>Logic Level<br>(binary) | Expected<br>Logic Level<br>(binary) |
|-------------------------------|--|-------------------------------------|
| 0.0                           | 0  | 0                                   |
| 0.2                           | 0  | 0                                   |
| 0.4                           | 0  | 0                                   |
| 0.6                           | 0  | 0                                   |
| 0.8                           | 0  | 0                                   |
| 1.0                           | 0  | -                                   |
| 1.2                           | 0  | -                                   |
| 1.4                           | 0  | -                                   |
| 1.6                           | 1  | -                                   |
| 1.8                           | 1  | -                                   |
| 2.0                           | 1  | 1                                   |
| 2.2                           | 1  | 1                                   |
| 2.4                           | 1  | 1                                   |
| 2.6                           | 1  | 1                                   |
| 2.                            | 1  | 1                                   |
| 3.0                           | 1  | 1                                   |
| 3.2                           | 1  | 1                                   |

Note: The data is valid for resistive loads terminated to both VDD and GND. Blank values in the expected logic level column indicate illegal conditions.

The loads were then reconfigured to be capacitive loads terminated to ground, and the tests were run again. The results are shown in table E-3. The observed signal glitching is an unexpected and unsafe outcome. For input voltages between 2.0 and 2.4 V, the outputs changed, even though the signal met or exceeded the V<sub>IH</sub> requirement. This indicates the LVTTL standard has been violated. Clearly, capacitive loads can produce an erroneous circuit operation.

Table E-3. Logical Output Voltages as a Function of Input Voltage of the Nonswitching Input for Capacitive Loads

| Vin_dc<br>(Volts) | test_out<br>Logic Level<br>(binary) | Expected<br>Logic Level<br>(binary) |
|-------------------|-------------------------------------|-------------------------------------|
| 0.0               | 0                                   | 0                                   |
| 0.2               | 0                                   | 0                                   |
| 0.4               | 0                                   | 0                                   |
| 0.6               | 0                                   | 0                                   |
| 0.8               | 0                                   | 0                                   |
| 1.0               | 0                                   | -                                   |
| 1.2               | 0, with<br>glitches                 | -                                   |
| 1.4               | Severe<br>Glitching                 | -                                   |
| 1.6               | Severe<br>Glitching                 | -                                   |
| 1.8               | 1, with<br>glitching                | -                                   |
| 2.0               | 1, with<br>glitching                | 1                                   |
| 2.2               | 1, with<br>glitching                | 1                                   |
| 2.4               | 1, with<br>glitching                | 1                                   |
| 2.6               | 1                                   | 1                                   |
| 2.8               | 1                                   | 1                                   |
| 3.0               | 1                                   | 1                                   |
| 3.2               | 1                                   | 1                                   |

Notes: Data valid for capacitive loads terminated to both VDD and GND. Also, the shading indicates error conditions.

### E.5.3 TEST CASE 1 EXTENSION—USING THE ILA TO VERIFY DESIGN FUNCTIONALITY.

The Xilinx Chipscope ILA was used to perform the SSN experiments. To ensure that the SSN errors reported by the ILA were real, a hardware failure indication method was implemented in the FPGA. This hardware included two latches: one latch would turn on an LED, LED1, when a high signal was detected on signal `dc_in`, and the other latch would turn on an LED, LED2, when a low signal was detected on signal `dc_in`.

To perform this experiment, the S3 load board was configured with 47-pF loads and  $V_{term} = GND$ . `Dc_in` was set to 0 V, and the latches were cleared. The voltage on `dc_in` was increased until LED1 turned on, indicating a logical high on signal `dc_in`. This occurred at  $V_{dc\_in} = 0.84$  V, indicating an error due to SSN. The ILA did not detect any errors for this test condition until  $V_{dc\_in} = 1.1$  V. This experiment was then performed starting with  $V_{dc\_in} = VDD$ , and decreasing  $V_{dc\_in}$  until LED2 turned on. A transition was observed at  $V_{dc\_in} = 2.93$  V, indicating an SSN error. The ILA did not detect any errors for this test condition until  $V_{dc\_in} = 1.5$  V.

The above experiment confirms that the errors found by the ILA were real errors. This experiment also presents another potential safety issue regarding the use of AEH tools—integrated hardware debugging tools, such as ILA, logically probe the internal FPGA signals. There is a possibility that measurements provided by the tool may not match the actual results observed on the external signals.

### E.5.4 TEST CASE 1—ANALYSIS.

A standard FPGA was examined to observe if simultaneously switching outputs would introduce errors in FPGA operation. The number of I/Os that were switched simultaneously were chosen to be within the manufacturer's recommendation. These I/Os were then connected to resistive loads consistent with the LVTTL specification. Measurements of the FPGA under resistive load conditions did not find any undesirable circuit operation. The I/Os were then connected to capacitive loads consistent with the LVTTL specification and the measurements were repeated. In this case, glitching was observed for inputs between 1.2 and 2.4 V. Since glitching signals can cause errors in other circuits, this circuit was unsuitable for safety-critical operation. In addition, glitching occurred at conditions that met the LVTTL standard; therefore, the FPGA was operating in violation of the standard.

SSN effects were observed to be large enough to produce an erroneous circuit operation for capacitive loads, but not for resistive loads. This happened despite the fact that the magnitude of the currents drawn in the resistive and capacitive load cases was similar. At first glance, one would expect similar circuit performance for similar current loads. However, SSN was proportional to the derivative of the current and not the magnitude of the current. When switching initially occurred, the capacitor appeared to the output as a short, and this caused a high, instantaneous current and a correspondingly high  $di/dt$ . Because of this effect, capacitive loads were far more likely to produce SSN-related circuit errors.

### E.5.5 CONCLUSIONS.

Erroneous circuit operation occurred when the outputs were connected to capacitive loads and switched simultaneously. The circuit itself operated entirely as intended. It is unlikely that the additional design verification processes identified in appendix B of DO-254 would identify this failure mechanism. The erroneous glitching depends on the current data pattern that is presented to the circuit as well as on previous data patterns. The root cause of the error is parasitic inductances and resistances that are normally not considered in complex hardware. SSN introduces noise that varies much more slowly than the data rate. Whether or not the noise produces errors depends on the timing of the noise peaks and the data. This timing is highly dependent on the sizes of the parasitic elements in the package and in the layout of the printed circuit board to which the component is connected. Simulating and analyzing a full design while considering all of the parasitic elements is computationally infeasible at this time.

Although errors were observed when switching the maximum capacitive load, additional research is needed to determine the maximum capacitive load that can be safely switched. Because glitching can introduce errors in other parts of the circuit, SSN can be a concern for safety-critical hardware when driving purely capacitive loads, even if a violation of the LVTTTL standard does not occur. It may be possible to alleviate the problems occurring when driving capacitive loads by adding resistive terminations to the loads, but this condition was not analyzed.

### E.6 TEST CASE 2—UNDEFINED I/O STATES.

Unused FPGA I/O pins can have residual logic, be grounded, be active, or be floating. Ultimately, the tool determines what happened to the pins left undefined in a design. The method by which a tool chooses to handle these unused pins is a safety concern.

The code will be written so that a signal of variable frequency from zero to the maximum frequency of the FPGA gate logic can be routed through a significant portion of the components of the FPGA, and then connected to a significant portion of the I/O pins. At least a single path will be coded with a known logic outputting to the I/O pins. Figure E-10 is an adequate representation for this purpose; except, in this case, the solid lines represent previously programmed logic. The design will be programmed into the FPGA following an implementation that used a vast majority of gate count, such as in the power integrity case.

Two signal sources are required. The constant signal path will be a fixed square wave of known input and output. The frequency input will be a 100-Hz signal; the output can be determined by the design of the ripple counter. The controllable signal path will be a sweep from 0 to 2 kHz.

A significant amount of previously assigned I/O pins will be analyzed with an oscilloscope to determine if they are floating or fixed by probing I/O pins. A 10k- $\Omega$  resistor is used to pull the solid pins first to VDD and then to GND to measure and record the pin voltage. The previously programmed paths will be monitored with a logic analyzer while sweeping the frequency. Simultaneously, all the known logic paths will be monitored.

### E.6.1 TEST CASE 2—TEST PROCEDURE.

The undefined I/O test case is used to determine if previously loaded programs can affect the undefined I/O pins. The test will begin by loading a known hardware implementation (the SSN test) that uses pins that are unused in the test implementation. Then the test implementation will be loaded, and the undefined I/O pins are examined to see if there is any residual logic connected to these pins.

The state of the unused I/O will be determined by observing the I/O behavior using an oscilloscope under the following conditions:

- The I/O will be pulled to VDD through a 10k- $\Omega$  resistor. If the I/O voltage measured is equal to ground, the I/O is being actively driven low. If the I/O voltage is equal to VDD, the I/O is either being actively driven high or it is in a high impedance state.
- The I/O will be pulled to ground through a 10k- $\Omega$  resistor. If the I/O voltage measured is equal to VDD, the I/O is being actively driven high. If the I/O voltage is equal to GND, the I/O is either being actively driven low or it is in a high impedance state.

If the I/O toggles, the signal is being driven by logic in the FPGA. For this experiment, the load board is populated with 10k- $\Omega$  loads on the unused I/O. Jumpers Vterm1 and Vterm2 can then be used to pull the unused I/O to VDD or GND.

### E.6.2 TEST CASE 2—UNDEFINED I/O TEST RESULTS.

Two variations of the test were run. In the first variation, the signal toggle\_out was removed from both the user constraint file (UCF) and the top-level Verilog<sup>®</sup> module. The behavior of the unused I/O was then examined with the loads terminated to both VDD and GND. The results from this experiment are shown in table E-4.

Table E-4. Unused I/O Voltage Levels

| FPGA Pin Number | VIO         |             | FPGA Pin Number | VIO           |               |
|-----------------|-------------|-------------|-----------------|---------------|---------------|
|                 | Vterm = VDD | Vterm = GND |                 | Vterm = VDD   | Vterm = GND   |
| D5              | 0.312 V     | 0 V         | E6              | 0.306 V       | 0 V           |
| D6              | 0.307 V     | 0 V         | C5              | 0.307 V       | 0 V           |
| E7              | 0.311 V     | 0 V         | C6              | 0.311 V       | 0 V           |
| D7              | 0.306 V     | 0 V         | C8              | 0.311 V       | 0 V           |
| D8              | 0.312 V     | 0 V         | A3              | 0.310 V       | 0 V           |
| B4              | 0.306 V     | 0 V         | A4              | 0.310 V       | 0 V           |
| B5              | 0.309 V     | 0 V         | A5              | 0.309 V       | 0 V           |
| B6              | 0.312 V     | 0 V         | B7              | 0.310 V       | 0 V           |
| A7              | 0.312 V     | 0 V         | B8              | Toggle output | Toggle output |
| A8              | 0.309 V     | 0 V         | B13             | Echo DC in    | Echo DC in    |

The results of this experiment indicate that strong internal pull-down resistors are implemented on the FPGA on all unused I/Os. The Spartan-3E family datasheet states that the default configuration for unused I/O is an internal pull-down resistance, confirming the results of this experiment. The unused I/O can be optionally configured with internal pull-up resistance or with no internal resistors (high impedance).

In the second variation, the signal `toggle_out` was removed from the UCF, but the top-level Verilog module was left unchanged. When building this test design, the AEH tool considered these signals to be external signals. Since these signals were not specified in the design constraints, the AEH tool automatically selected which FPGA I/O pins these signals would be connected to. This presents a potential risk to FPGA designs. If an FPGA hardware description language design contains ports that are not actually used by hardware, the AEH tool may still connect these ports to external FPGA pins. These pins would be unexpectedly driven by logic in the FPGA.

In the above test, the AEH tool did generate a warning about this problem in the place-and-route report. The warning stated that a partially locked I/O bus was found in the design. By viewing the pad report (the report the tool produces), which contains information on how the FPGA I/Os are implemented, the pins to which the unspecified I/Os were connected can be found.

### E.6.3 TEST CASE 2—UNDEFINED I/O TEST CONCLUSIONS.

The first test found that unused pins are connected to a default pull-down resistor. No residual logic was found. The second test showed that there is a risk of unknowingly driving external FPGA I/Os with internal FPGA logic. However, these problems can be avoided by constraining all I/O signals in an FPGA project to known pin locations. Furthermore, any I/O pins that are intended to be unconnected in an FPGA design can be constrained as prohibited locations. In a Xilinx<sup>®</sup> design, “CONFIG PROHIBIT=location;” is the prohibit constraint. This constraint prevents the accidental use of the specified pin.

## E.7 TEST CASE 3—TIMING CONSTRAINTS.

The purpose of this case study is to determine if a circuit meets the real-time constraints that the tool displays in the design report and that any designer-imposed timing margins are maintained. The tools have the ability to specify the time that it will take a given operation to complete. If the bounds of the speed of the gates are pushed close to their extremes, the tool will redesign the circuit so that the delay is smaller. The following questions are of interest: Where are these bounds? How close does the tool allow the design to get to the bounds? Is there a safety margin with actual delay and estimated delay?

The case study will be implemented with an asynchronous ripple counter. As shown in figure E-14, an asynchronous ripple counter is a series of N flip-flops cascaded together so that the output of a flip-flop is the clock input of the next one. The input to the ripple counter will be an external pin that is connected to a high-frequency source. The final output of the counter will be connected to an output pin. The expected output signal frequency will be the input signal frequency divided by  $2^N$ . During the initial implementation, a report is generated by the tool to establish the maximum input frequency and the signal path delay. All signals will be transistor/transistor logic.

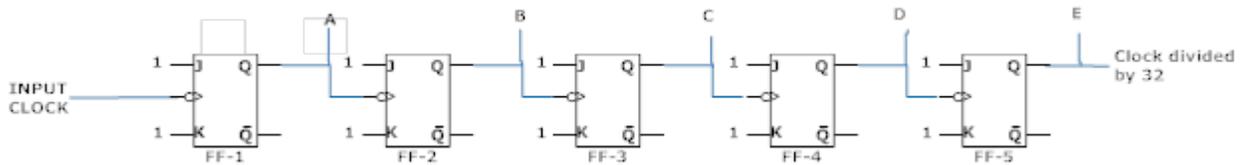


Figure E-14. Ripple Counter

Timing constraints will be set for the ripple counters, and a timing report will be generated by the tool. The design will be synthesized and then tested to observe if the timing constraints were accurate and provided adequate margin.

During the sweep, both the input and output wave of the component will be scoped for accuracy and phase differences. The input frequency will be increased until either the signal path time does not meet requirements or the output wave is not correct. The phase differences will yield the signal path time; the point of failure will determine the safety margins applied. All data and settings will be recorded accurately for analysis.

### E.7.1 TEST CASE 3—IMPLEMENTATION DETAILS.

Test Case 3 was used to evaluate the accuracy of timing constraints in an FPGA design. Project AEH\_case3a is the Verilog implementation of this test case. The AEH\_case3a design consists of an input clock signal, `clk_in`, a frequency generator that provides signal `fast_clk`, and an 80-bit counter that is incremented by signal `fast_clk`. The size of the counter was chosen because it allowed for a maximum operating frequency in the range of 125 MHz.

Counter bits 15 and 0 were connected to FPGA I/O C6 and B12, respectively, so that they could be monitored on an oscilloscope through the load board SMA connections. Counter bit 79 was connected to FPGA I/O E6. This signal, the most significant bit of the counter, was connected to an output simply because, if it was not used, the AEH tool recognized and removed the unused logic associated with this signal.

An 8-ns period (125-MHz) timing constraint was specified for this design, meaning that the counter was able to properly function at up to 125 MHz. The post place-and-route timing analysis of the design indicated a maximum allowable frequency of 125.5 MHz, which slightly exceeded the timing constraint.

The counter outputs were monitored using ILA to detect errors, while the frequency of fast\_clock was set to various frequencies. The clock frequency was first set at 100 MHz, and the design performance was evaluated. The clock frequency was then incrementally increased until the point at which errors were observed on the counter outputs. Of primary interest was the operation of the FPGA system at or near the frequency limit specified by the FPGA design tool; the presence of any errors at or below this frequency was considered a timing constraint failure. If no errors were observed at or below the maximum specified clock frequency, the clock frequency was increased until errors were observed. This demonstrated the amount of margin included in the design by the AEH tool.

### E.7.2 TEST CASE 3—RESULTS.

The purpose of Test Case 3 was to evaluate the accuracy of timing constraints in an FPGA design. This was examined by implementing an 80-bit counter in an FPGA and operating the counter at various frequencies. The least significant 20 bits of the counter were captured for 8192 samples using ILA (figure E-15), and the output data were analyzed for errors using a spreadsheet (figure E-16). The output data were checked for errors using the following formula:  $\text{Error} = \sum (\text{abs}((a_n - a_{n-1}) - (a_{n-1} - a_{n-2})))$ ,  $n = 2$  to  $n = 8191$ , where  $\text{Error} \neq 0$  indicates an error.

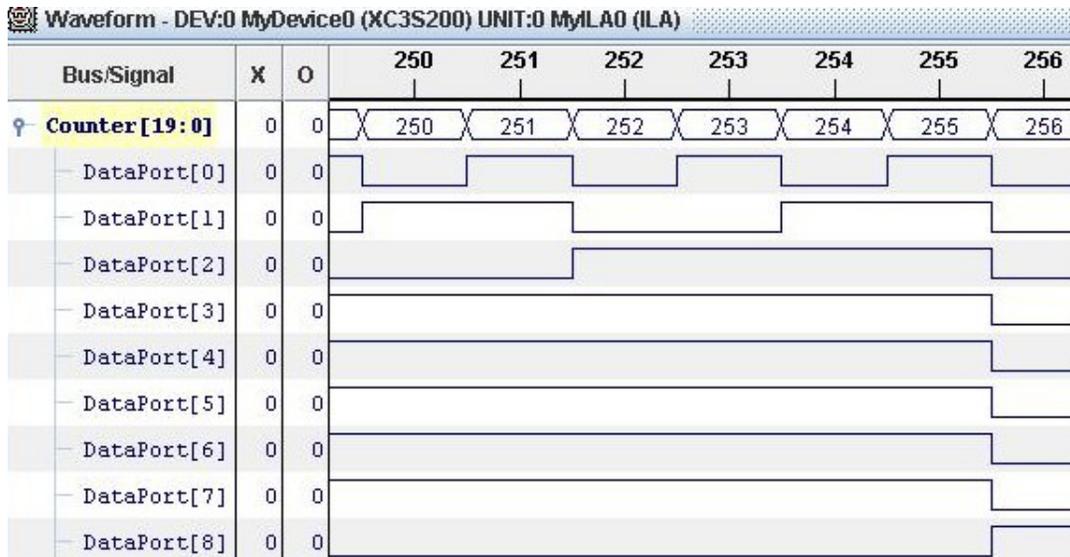


Figure E-15. An ILA Capture of the Counter Outputs

| A                | B             | C          | D           | E               | F     |
|------------------|---------------|------------|-------------|-----------------|-------|
| Sample in Buffer | Counter[19:0] | difference | expect zero | abs expect zero | error |
| 0                | 0             | 1          | 0           | 0               | 0     |
| 1                | 1             | 1          | 0           | 0               | 0     |
| 2                | 2             | 1          | 0           | 0               | 0     |
| 3                | 3             | 1          | 0           | 0               | 0     |
| 4                | 4             | 1          | 0           | 0               | 0     |
| 5                | 5             | 1          | 0           | 0               | 0     |
| 6                | 6             | 1          | 0           | 0               | 0     |
| 7                | 7             | 1          | 0           | 0               | 0     |
| 8                | 8             | 1          | 0           | 0               | 0     |
| 9                | 9             | 1          | 0           | 0               | 0     |
| 10               | 10            | 1          | 0           | 0               | 0     |
| 11               | 11            | 1          | 0           | 0               | 0     |
| 12               | 12            | 1          | 0           | 0               | 0     |
| 13               | 13            | 1          | 0           | 0               | 0     |
| 14               | 14            | 1          | 0           | 0               | 0     |
| 15               | 15            | 1          | 0           | 0               | 0     |
| 16               | 16            | 1          | 0           | 0               | 0     |

0 indicates no errors

Figure E-16. Error Tabulation Spreadsheet Showing No Errors Found

Table E-5 shows the frequencies that were tested and the maximum operational frequency specified by the AEH tool for the particular FPGA build. Because an internal FPGA clock multiplier was used, each test frequency required a new FPGA build, which is why different maximum operational frequencies are shown for the different builds.

Table E-5. Test Frequencies and the Associated Timing Constraints From the AEH Design Tool

| Test Frequency (MHz) | Max Guaranteed Frequency (MHz) | Timing Comments   | Results            |
|----------------------|--------------------------------|---|--------------------|
| 75                   | 122                            | Design meets timing   | No errors detected |
| 100                  | 117                            | Design meets timing   | No errors detected |
| 115                  | 119                            | Design meets timing   | No errors detected |
| 120                  | 123                            | Design meets timing   | No errors detected |
| 125                  | 126                            | Design meets timing   | No errors detected |
| 130                  | 127                            | Design does not meet timing   | No errors detected |
| 135                  | 126                            | Timing impossible warning. Set AEH tool to ignore timing constraints. | No errors detected |
| 140                  | 126                            | Timing impossible warning. Set AEH tool to ignore timing constraints. | No errors detected |
| 150                  | 126                            | Timing impossible warning. Set AEH tool to ignore timing constraints. | No errors detected |
| 175                  | 126                            | Timing impossible warning. Set AEH tool to ignore timing constraints. | No errors detected |

The AEH tool indicated that designs for test frequencies 75 to 125 MHz met the specified timing constraints. The AEH tool indicated a timing constraint failure occurred for test frequency 130 MHz. At frequencies above 130 MHz, the AEH tool indicated that it was impossible for the design to meet the specified timing constraints. To generate these higher-frequency FPGA builds, the AEH tool had to be configured to ignore timing constraints. At all of the test frequencies, no operational failures were detected in these experiments. At 175 MHz, the maximum frequency tested, the ILA was reconfigured to examine the 20 most significant bits of the counter. For this variant, no errors were detected.

### E.7.3 TEST CASE 3—TIMING CONSTRAINT ACCURACY CONCLUSIONS.

The timing constraint accuracy experiments did not detect any timing-related failures in the FPGA. Although the AEH tool indicated that the design speed was limited to approximately 125 MHz, the counter operated successfully at much higher speeds. Although this design operated well above the specified frequency limit, the timing constraints specified by the AEH tool should be followed. The AEH tool may allow for some timing margin beyond the specified constraints, but this margin will vary based on the specific application.

## E.8 EXPERIMENT CONCLUSIONS.

Test case 1 examined simultaneous switching noise (SSN) effects, which could potentially cause hardware failures when numerous field-programmable gate array (FPGA) input/output (I/O) switch at the same time. This experiment was performed when switching multiple outputs into both resistive and capacitive loads. The experiments did not indicate any failures when driving resistive loads, indicating that the FPGA can meet the simultaneous switching I/O specifications, even when driving the maximum specified direct current output current.

When driving capacitive loads, a large I/O current is produced during switching. Under this condition, failures were observed due to SSN. When driving numerous I/O into capacitive loads, SSN can present a serious risk to systems using FPGAs.

As an extension to test case 1, an experiment was performed to determine whether or not the FPGA Integrated Logic Analyzer (ILA) reported the same hardware performance as external hardware measurements. It was found that hardware errors can occur external to the FPGA and not be reported by the ILA. This presents an additional airborne electronic hardware (AEH) risk for designs using the FPGA internal hardware debugging tools.

Test case 2 examined the behavior of unconnected I/O in an FPGA, which could potentially be configured in an unknown or unpredictable way. Of particular concern was whether or not unused I/O may contain residual logic or be actively driven. The experiments showed that unused I/Os are configured in a known manner, and risks associated with unused I/O can be avoided. Although unused I/Os are configured in a known manner, this configuration can consist of either internal pull-up resistors, internal pull-down resistors, or no internal terminations. For external hardware that is connected to an FPGA but not implemented in the FPGA, the chosen configuration mode could potentially cause the external hardware to behave in an unexpected way. The FPGA design engineer must consider the state of unused I/O and choose a configuration option that is compatible with all external hardware connected to these pins. If the designer is unaware of what to do with these pins, then the hardware requirements are incomplete. An additional potential risk identified in test case 2 is that when a design contains external ports without specific I/O constraints, the AEH tool may automatically route these signals to I/O pins. This could cause an I/O pin to unexpectedly be driven with internal FPGA logic. This problem can be avoided by ensuring that all I/O signals in an FPGA project are constrained to specific I/O pins.

Test case 3 examined the accuracy of timing constraints in FPGA designs. The AEH development tools perform timing analysis on FPGA designs. The purpose of this experiment was to determine the accuracy of this timing analysis—specifically, can the design operate up to the specified frequency limit and, if so, how much operational margin is provided beyond the specified frequency limit? The timing constraint accuracy experiments did not detect any timing-related failures in the FPGA. Additionally, the experiments showed that the design could operate at speeds well above the limit specified by the AEH tool. It is likely that this is implementation-specific, and that the amount of operational margin beyond the specified maximum frequency will depend on the particular application.

## E.9 REFERENCES.

- E-1. Xilinx Corporation Spartan-3E Family Data Sheet, December 4, 2009, pp. 13, [http://www.xilinx.com/support/documentation/data\\_sheets/ds099.pdf](http://www.xilinx.com/support/documentation/data_sheets/ds099.pdf).
- E-2. Xilinx Corporation Spartan-3E Family Data Sheet, December 4, 2009, extracted from pp. 3, [http://www.xilinx.com/support/documentation/data\\_sheets/ds099.pdf](http://www.xilinx.com/support/documentation/data_sheets/ds099.pdf).
- E-3. Xilinx Inc., “Spartan-3E FPGA Family: Data Sheet, DS312,” August 26, 2009, pp. 136-137.

## APPENDIX F—EVALUATION REPORT FOR HARDWARE DESIGN TOOLS

### F.1 PRODUCT DESCRIPTIONS.

The following section provides a short description of the three leading hardware design tools.

#### F.1.1 QUARTUS® II.

Quartus II design software delivers the highest productivity and performance for Altera® field-programmable gate arrays (FPGA), complex programmable logic devices (CPLD), and HardCopy® application-specific integrated circuits, and offers numerous design features to accelerate the design process:

- Design entry
- Scripting support
- Incremental compilation: initial setup
- System on a programmable chip builder
- MegaWizard® Plug-In Manager
- I/O Pin Assignment Analysis
- Quartus II Integrated Synthesis
- Third-party design entry and synthesis
- Basic compilation flow

The version of software tested was Quartus II Version 7.0 by Altera Corporation, 101 Innovation Drive, San Jose, CA 95134. It is used for the Altera DE2 Development and Education Board.

#### F.1.2 XILINX® ISE®.

Xilinx offers the ISE Design Suite with SmartCompile™ Technology for faster programmable logic device timing closure and maximum performance, as well as a range of optional products that deliver unprecedented designer productivity. Xilinx ISE features include:

- Breakthrough performance with ISE Fmax Technology
- Faster, easier timing closure with SmartCompile Technology
- SmartXplorer to provide distributed processing for more turns-per-day
- Advanced verification and power analysis
- PinAhead technology to simplify the complexities of FPGA pin assignment
- An integrated, front-to-back design environment

The version of software tested was Xilinx ISE Design Suite 10.1 by Xilinx Inc., 2100 Logic Drive, San Jose, CA 95124-3400. It is used for the Digilent Spartan™-3E Starter Board.

### F.1.3 LabVIEW® 8.5.

LabVIEW is a graphical programming environment used to develop sophisticated measurement, test, and control systems using graphical icons and wires that resemble elements of a flow chart. LabVIEW offers integration with thousands of hardware devices and provides hundreds of built-in libraries for advanced analysis and data visualization. The LabVIEW platform is scalable across multiple targets and operating systems and, since its introduction in 1986, has become an industry leader in applications such as data acquisition, instrument control, measuring and controlling industrial systems, and embedded systems design. More recently, LabVIEW has been equipped with a capability for FPGA design. This feature provides an interesting alternative to traditional text-based design languages, such as VHDL and Verilog®, since it used a completely new graphical approach to programming.

The version of software tested was LabVIEW 8.5 by National Instruments™ Corp, 11500 N Mopac Expwy, Austin, TX 78759-3504. It is used for the National Instruments (NI) Compact Reconfigurable I/O (cRIO) 9074 integrated system.

### F.2 EVALUATION PLAN.

This evaluation used a variation of the process described in reference F-1. Three software tools, used for FPGA development, were evaluated:

- Quartus II by Altera®
- ISE by Xilinx
- LabVIEW FPGA module by NI

The Quartus II software was used to compile VHDL code and upload this code to an Altera DE2 Development and Education Board for execution on an Altera Cyclone® II FPGA. The Xilinx ISE was used to compile VHDL code and upload this code to a Digilent Spartan-3E Starter Board for execution on a Xilinx Spartan-3E FPGA. The LabVIEW FPGA module was used to develop and upload LabVIEW Virtual Instruments to the reconfigurable FPGA within the NI cRIO-9074 integrated system.

Experiments performed on these software tools were conducted in two phases. The first phase was to facilitate learning and familiarization of the tools by using a simple “Hello World” type program, executed on the development boards. The second phase was to design an up-down counter using the tools, also executed on the development boards.

This report summarizes the test effort. Any variance from the test plan and the reason for the variance is recorded. Abnormal termination and any unresolved test incidents are recorded in the summary of results section of the test summary report. Differences between the manufacturer’s specifications and the test results for Quartus II and the Xilinx ISE Design Suite are recorded. The final product of this evaluation is a test summary report.

### F.2.1 EVALUATION TEAMS AND CHARTER.

The evaluation team was composed of the following members:

- Joseph Voelmle, student—Responsible for designing and conducting all tests, compiling test results, and writing evaluation report.
- Dr. Janusz Zalewski, faculty member—Supervised the evaluation, comments on the report, defined the experiments, and interpreted the results.

Both team members derived evaluation criteria. Before the evaluation began, the following questions were posed:

- What is the evaluation expected to achieve? Determine the functionality, usability, and efficiency of Altera Quartus II, Xilinx ISE, and NI LabVIEW FPGA module.
- What are the responsibilities of each member of the team? Joseph Voelmle will be responsible for conducting all tests, and Dr. Janusz Zalewski will provide supervision and technical advice. Both team members will derive evaluation criteria.
- How will success be measured? What tasks are to be performed to measure desired metrics?
- What is an exit criterion for evaluation? When all desired tasks are measured.
- What constraints must the evaluation team adhere to? The team must use the same exact procedures to evaluate all tools.
- Goals of the evaluation—What are the criteria and what is their usefulness in evaluating hardware design tools applied in safety-critical systems?
- What is the scope of the evaluation?

### F.2.2 APPROACH TO EVALUATION.

There are a number of factors that may affect system safety. For instance, Dahll, et al. [F-2 through F-4] list the following: system quality, complexity, user experience, fault tolerance, producer's pedigree, documentation, testing, quality assurance, tool quality, and other factors. One of the factors shown in figure F-1 is tool quality and is discussed in more detail herein.

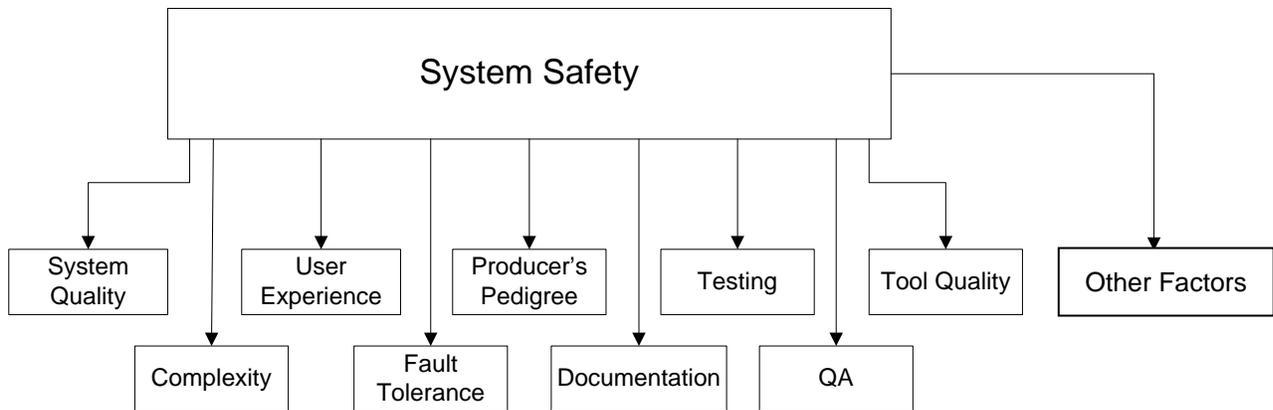


Figure F-1. Factors Affecting System Safety

To evaluate tool quality completely, one would need to look at it from three different perspectives, and collect data accordingly:

- How the tool itself was developed.
- How the tool is operating.
- How much quality this tool built into the product.

The framework for this process, based on the context of tool use, is shown in figure F-2. The previous work on tool evaluation [F-1] states:

“The central part of this model is the *macroevaluation* based on the use of the tool during the design phase. However, much information on tool quality can be derived from the development of the tool itself, considered as a *metaevaluation*: evaluating the process to develop a tool. The tool vendor can provide the data for evaluation of this stage. In addition to the *macro-* and *metaevaluation*, the product developed with a particular tool can be included in the evaluation. This is called *microevaluation*, and it focuses on the level lower than the tool itself. Such a product evaluation can be based both on static code analysis and code execution. Consequently, to have the entire picture of the tool’s quality, one needs to do the evaluation at three different levels.”

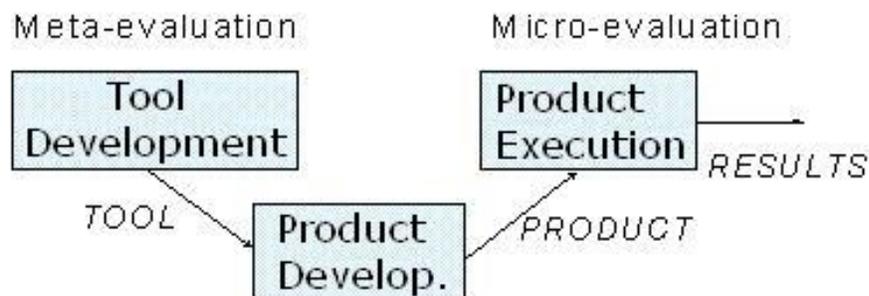


Figure F-2. Model of the Tool Evaluation Process

However, for various reasons, it is next to impossible to obtain data on tool development from the tool vendors. This is mainly due to the vendor's reluctance to release proprietary information to the public where it could possibly be used by competitors. For this reason, performing the meta-evaluation is normally not done. Therefore, this work focuses on macro- and microevaluation.

To evaluate tool quality as a system safety factor, the following three criteria were chosen: functionality, usability, and efficiency, according to a previous work [F-1]. These evaluation criteria were used to measure the quality of the Altera Quartus II Design Suite, Xilinx ISE Design Suite, and NI LabVIEW FPGA hardware design tools. This was approached in two steps for each tool. The first step was to become familiar with the tools and learn their capabilities. The second step was to use the knowledge gained from the first step to design a simple VHDL circuit to evaluate the tools from a designer's perspective. The evaluations were performed in the following four steps, referred to as tasks in previous work related to software development tool evaluations [F-1]:

1. Project preparation and tool familiarization
2. Model development and code generation
3. Measurement and data collection
4. Postmortem, including data analysis and report generation

Two sample problems were used to evaluate model development and code generation. The first was a "Hello World"-type program in order to cause a light-emitting diode (LED) on the evaluation board to blink at a specified rate. The VHDL code is presented in section F.8. This program was derived from a tutorial by Martin Schoeberl [F-6]. The second problem was a simple up-down counter program. Its VHDL code is shown in section F.9. Its purpose was to gauge the user's level of comprehension and familiarity of the tools once basic mastery was accomplished with the Hello World program. The approach was to use identical code for all three software tools, so that a useful comparison could be made.

The ultimate goal of this approach to software tool evaluation was to determine if the results could be used in the next stage, or project, for a more extensive evaluation on a real-life project.

### F.3 EVALUATION CRITERIA.

#### F.3.1 DEFINITIONS.

The following definitions of basic measurement concepts were adopted from engineering publications such as in reference F-7.

- Efficiency—A property determining the degree to which a system or component performs its designated functions with minimum consumption of resources [F-7]. In particular, this can be applied to execution efficiency and storage efficiency, which would be the speed and size of the code produced by the software.

- **Functionality**—The capacity of a computer program or application to provide a useful function. Functionality relates to “what” the user wants from the system. Only the user can evaluate this property of the software, and these attributes are very dependent on the nature of the project that the tool is helping to develop.
- **Measure**—A physical or abstract device that is used to apply a metric (i.e., a ruler).
- **Measurement**—The act or process of assigning a number or category to an entity to describe quantitatively a property of that entity. A figure, extent, or amount is obtained by a measurement.
- **Metric**—A scale, with a defined unit, that quantitatively characterizes a certain property (for example: an inch or centimeter to measure distance, thickness, etc.).
- **Software Quality Metric**—A function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which software possesses a given attribute that affects its quality.
- **Usability**—A property determining an ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component [F-7]. Usability is also a measure of interface quality that refers to the effectiveness, efficiency, and satisfaction with which users can perform tasks with a tool. There are multiple ways that usability can be measured. One would be the ease of use or user-friendliness. Another would be concerned with the features, (i.e., the presence or absence of certain features in the user interface such as windows, icons, and menus). A third measure of usability would be the operational feature capability of the software (i.e., how easily, effectively, and satisfactorily it can be used by specific users performing specific tasks, and in specific environments where usability is at the level of interaction between users and the artifact).

Figure F-3 shows the relationship between a specific property, that is evaluation criterion, a metric used for its evaluation, and a measure that is used as an evaluation device to quantify the metric. Figure F-4 shows how these concepts are applied to the tool evaluation, whose quality is one of the many factors that affect system safety as shown in figure F-1.

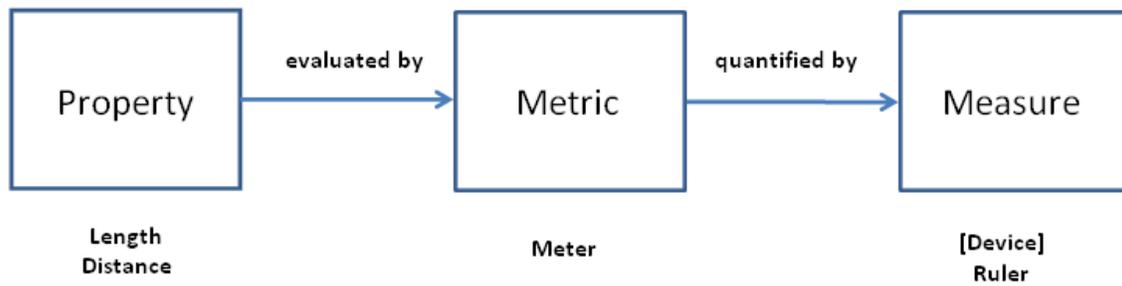


Figure F-3. Relationship Between a Property, Its Metric, and a Measure From the Point of View of Measurement Theory

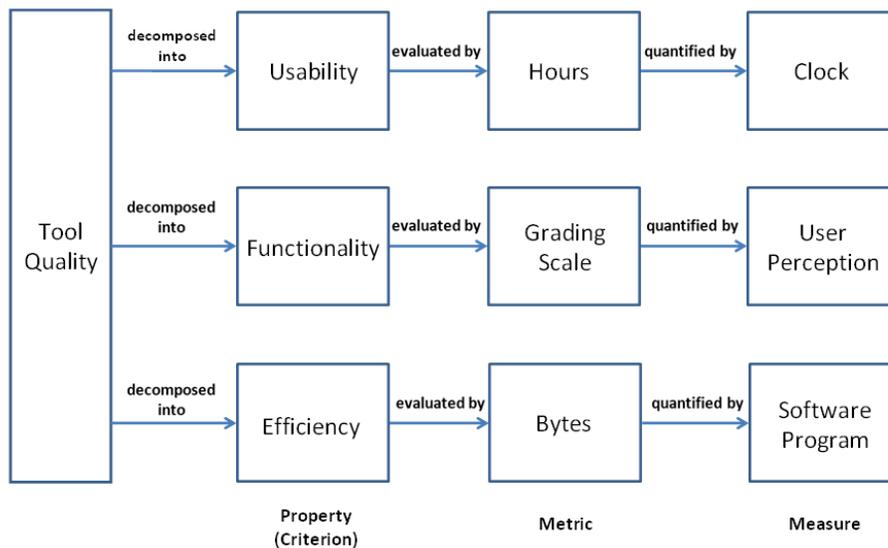


Figure F-4. Illustration of Tool Evaluation Concepts

### F.3.2 GENERAL METHODOLOGY.

The application of a measure to evaluate software properties has to be supported by a measurement method. A measurement method is a means for assessing and assigning a value to the product property, characterizing it quantitatively. This value can be obtained using a software quality metric, a unit of measure which is the standard of measurement that can be applied either by direct measurement or decomposition. For this evaluation, the following four steps (tasks) are used in decomposition:

1. Project preparation and tool familiarization
2. Model development and code generation
3. Measurement and data collection
4. Postmortem, including data analysis and report generation

In this evaluation, the properties measured—usability, functionality, and efficiency—can be determined only by decomposition steps. For example, one such metric for usability can be the

ease of learning how to operate Quartus II or Xilinx<sup>®</sup> ISE measured in hours spent learning the tools.

The data can be gathered on various scales, for example:

- Qualitative—Bad, fair, good (e.g., bad is unacceptable, fair is marginally acceptable, and good is fully acceptable).
- Quantitative—Hours spent.
- Rating scale—A scale of 0 to 5, with 5 being best, used as a subjective assessment.

Quantitative data were collected for the properties mentioned in the experiments described in section F.4. The functionality of the tools is measured using a rating scale of 0 to 5 for tutorial, user manuals, readability, and flexibility. In this report, only one developer (the tester) provided these ratings. With such a small sample to provide ratings, bias will obviously play a large role in a subjective measurement. However, it should be remembered that this evaluation is the basis for a larger, more extensive evaluation where there would be a much larger number of developers to provide ratings. Usability is measured as effort (in hours) spent on the four tasks listed in section F.2.2. Efficiency is measured in the code size generated by the tools for the sample code used.

#### F.4 METHODOLOGY APPLIED IN THIS RESEARCH.

The methodology used in this research was taken from reference F-1. As outlined in reference F-1, the following four tasks were performed to evaluate each tool:

- Project preparation and tool familiarization.
- Model development and code generation.
- Measurement and data collection.
- Postmortem, including data analysis and report generation.

Two experiments are performed to design a Hello World-type program and an up-down counter, as described in section F.4. The four tasks are performed on each experiment for each tool. Quantitative data is then collected for three criteria: efficiency, usability, and functionality. Efficiency is measured in code size generated by each tool. Usability is measured as time spent on the four tasks. Functionality is measured as a subjective assessment on a rating scale of 0 to 5.

#### F.5 DATA COLLECTION.

This section contains the actual results of measurements for both tools, the Altera Quartus II and the Xilinx ISE. The four tasks performed were those listed in section F.4.

The data were collected for the Hello World VHDL program and measured for usability, functionality, and efficiency for the Altera Quartus II. Once this data was collected, the next step

of the evaluation, using the Altera Quartus II to design an up-down counter, began. In this step, the Quartus II was measured with respect to usability, functionality, and efficiency performing the four tasks. Once these tasks were performed and data collected for Quartus II, they were repeated for the Xilinx ISE, again using the same Hello World and up-down counter programs to measure the criteria and collect data.

### F.5.1 QUARTUS II.

The Altera Quartus II design software provides a design environment that includes sample solutions for all phases of FPGA design. The Quartus II software consists of:

- Quartus II Design Suite
- MegaCore<sup>®</sup> IP Library
- Nios<sup>®</sup> II Embedded Design Suite
- ModelSim—Altera 6.1-g Web Edition

Once installed, Quartus II software's graphical user interface (GUI) is used to perform all stages of the design flow. Figure F-5 shows the Quartus II GUI as it appears when you first start the software.

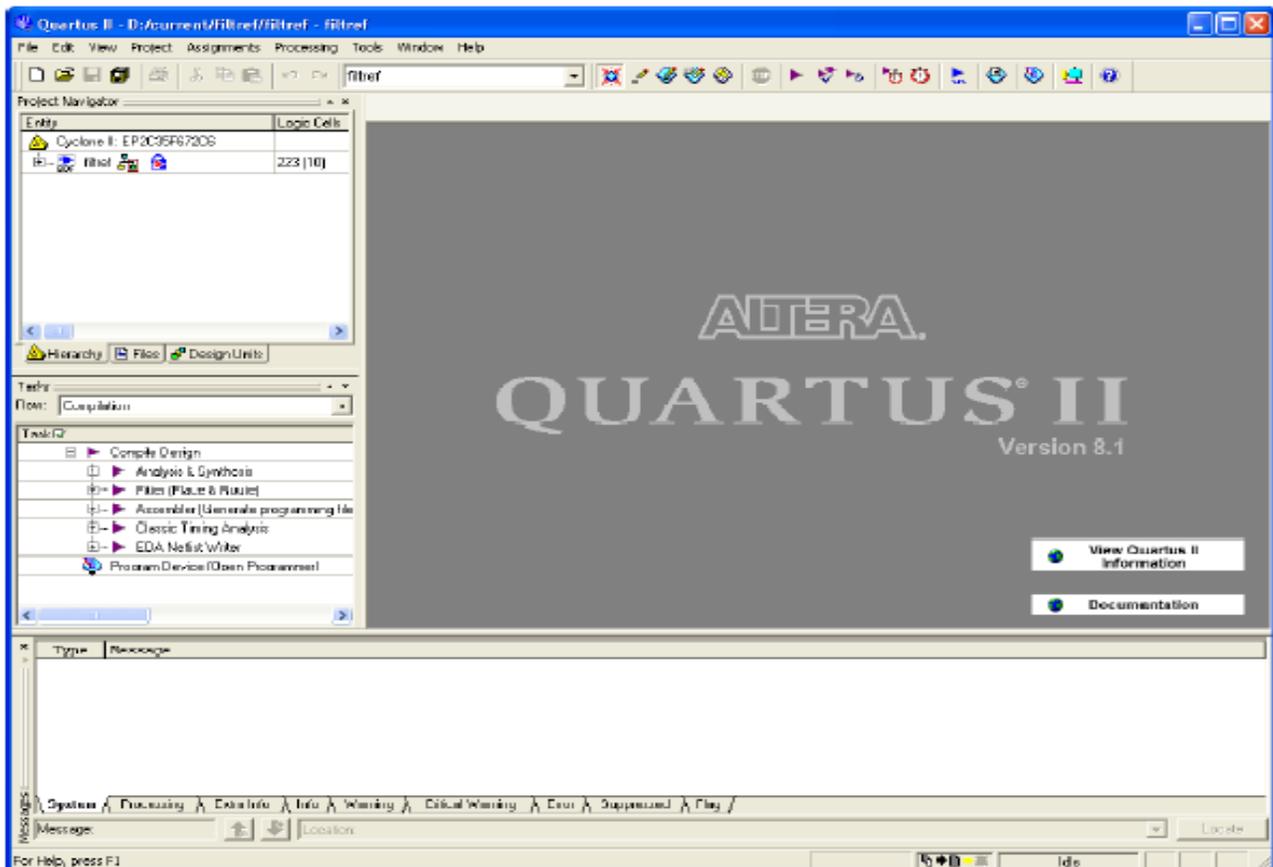


Figure F-5. Quartus II GUI [F-5]

The principle documentation for the novice Quartus II software user is the “Introduction to the Quartus® II Software.” As stated by Altera, the first two chapters give an overview of the major GUI, electronic design automation tool, and command-line interface design flows, with subsequent chapters leading the user through an overview of each task flow in the FPGA design. As such, the documentation does not state the intended audience for whom it is written or the level of expertise assumed for the user in the use of, and design for, FPGAs. The principle tester in this study had no previous experience with FPGAs. It was found that a tutorial with an example providing VHDL code was needed as a starting point. As there seemed to be no entry-level tutorial available from Altera, the tutorial from Martin Schoeberl [F-6] was used as a starting point. Problems soon occurred because it was discovered that the tutorial was written for a different version of the Cyclone II FPGA. The tutorial uses pin assignments that are different than the pin assignments needed for the Altera board. After further research, the correct pin assignments were located and substituted for the assignments in the tutorial. According to the tutorial, the files necessary to upload to the Altera board should be created, and the desired LED should blink. However, further problems were encountered as the desired files for FPGA were not created. This deviation from the tutorial is most likely due to the fact that a different FPGA is being used. A further search of the manufacturer’s website uncovered the instructions needed to create the required file and to have the desired LED blink.

#### F.5.1.1 Hello World.

For the Hello World program, the results of measuring the criteria are discussed in the following sections.

##### F.5.1.1.1 Usability.

The overall usability for Quartus II was measured at 90 hours. The usability for project preparation and tool familiarization took almost 60 hours. Much of this time was spent becoming familiar with the Quartus II software, including installing the Altera development board and learning about FPGAs in general. Also, time was spent becoming familiar with VHDL programming. A lot of time was spent consulting different sources to make the trivial Hello World program work. The manufacturer provides many manuals for the system, but they assume a level of sophistication that not all novice users may possess. It is the author’s opinion that time could have been saved if a basic, entry-level tutorial, specifically for this board, was available in one document. Since a program written by someone else was used, no time was spent developing it. However, approximately 16 hours were spent generating the code itself. Approximately 2 hours were spent measuring the tasks, which was done by logging all hours spent. Finally, the postmortem, where the data were collected and analyzed, took 12 hours.

##### F.5.1.1.2 Functionality.

In terms of functionality, the tester rated the following: tutorial—2, user manuals—4.5, readability—4.5, and flexibility—3.5. It is the author’s opinion that time could have been saved if a basic, entry-level tutorial, specifically for this board, was available in one document. The user manuals are high quality, providing detailed information in a logical and useful manner.

The model development and code generation functionality was rated 2.5. Functionality could have been higher if all the information needed to generate the code was in one place, and a search of the Internet was not needed.

#### F.5.1.1.3 Efficiency.

In terms of efficiency, the code generated 310 bytes of memory.

#### F.5.1.2 Up-Down Counter.

The next step was to perform the four tasks again using Quartus II to develop a simple up-down counter in VHDL code, and run it on the Altera board (see section F.9). Figure F-6 shows the schematic.

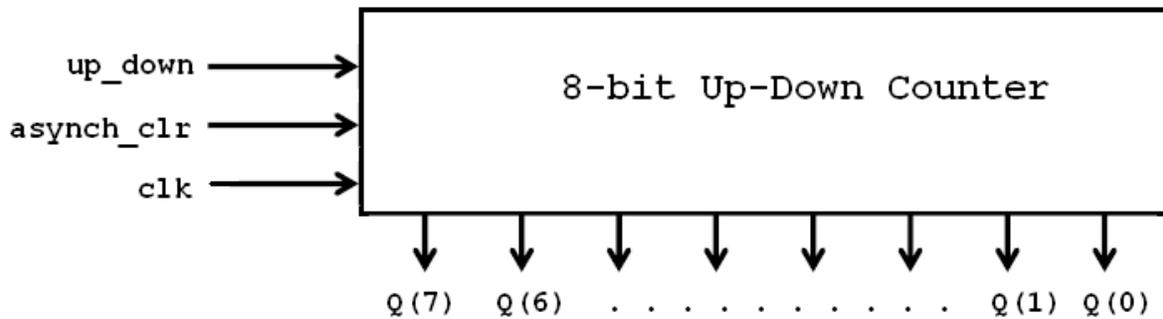


Figure F-6. Up-Down Counter

The counter has three inputs and eight outputs. The up-down input tells the counter in which direction to count, high voltage to count up and low voltage to count down. The asynch\_clr input resets the clock counter to zero. The counter will count up (or down) once with each clock at the clock (clk) input. The outputs, Q(0) through Q(7), are the 8-bit output of the counter, so on each clk pulse, the counter will proceed sequentially from (or down from) 00000000, 00000001, 00000010, 00000011, 00000100, up to 11111111 with each clock pulse. When it reaches 11111111, it goes to 00000000 on the next clock pulse and starts the sequence all over again.

The following sections discuss the same four tasks as those performed in the Hello World program and the results of measuring the criteria for the up-down counter.

#### F.5.1.2.1 Usability.

The overall usability for Quartus II was measured at 50 hours. The usability for project preparation and tool familiarization took almost 10 hours. Much of this time was spent becoming familiar with learning the VHDL language and how to apply it to designing an up-down counter. About 10 hours were spent learning new features of Quartus II and the Altera development board, which were not explored in the Hello World program. The bulk of the time was spent designing and writing the VHDL code, making corrections to the code due to compile

errors, and getting the up-down counter to run in the Altera board. As in the Hello World program, approximately 2 hours were spent measuring the tasks.

#### F.5.1.2.2 Functionality.

In terms of functionality, the tester rated the following: tutorial—2, user manuals—4.5, readability—4.5, and flexibility—3.5. Since the same documentation was used, the measured functionality was the same. The model development and code generation functionality was rated 3.5. The model development and code generation functionality was rated higher for the up-down counter because of the experience gained in the Hello World program and the user's confidence in using Quartus II.

#### F.5.1.2.3 Efficiency.

In terms of efficiency, the code generated 292 Bytes of memory.

### F.5.2 XILINX ISE.

The Xilinx ISE software provides a design environment that includes sample solutions for all phases of FPGA design. The Xilinx ISE software consists of:

- ISE Webpack™ 10.1
- ISE Foundation 10.1 Eval
- EDK 10.1 Eval
- ChipScope™ Pro 10.1 w/Serial I/O Toolkit License Key Eval
- System Generator AccelDSP™ Synthesis Tool 10.1 Eval
- PlanAhead™ Design Analysis Tool 10.1 Eval

Once installed, the Xilinx ISE software GUI is used to perform all stages of the design flow. Figure F-7 shows the Xilinx ISE iMPACT GUI.

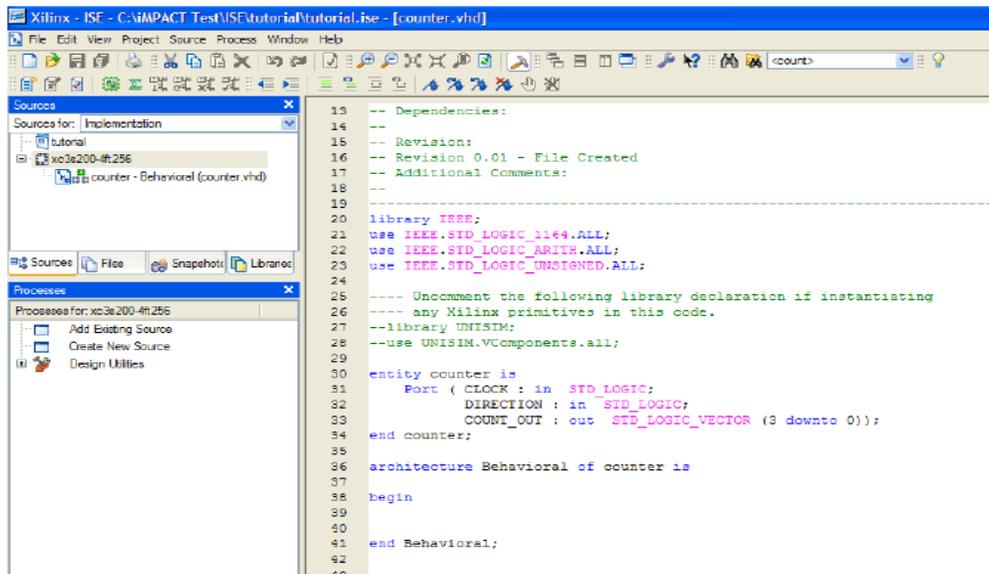


Figure F-7. Xilinx ISE iMPACT GUI [F-8]

The principle documentation for the novice Xilinx ISE software user is the “ISE 10.1 Quick Start Tutorial” [F-8]. In this tutorial, a counter is implemented. The tutorial was followed as written, except that the VHDL code of the Hello World program is substituted for the counter program in the tutorial. One thing that was noticed, which proved to be convenient, was that the pin numbers that need to be assigned are printed on the circuit board. For example, to use LED 0, assign F12 to the LED variable in the VHDL code. This was found to save time and effort compared to the Quartus II evaluation.

#### F.5.2.1 Hello World.

For the Hello World program, the results of measuring the criteria are discussed in the following sections.

##### F.5.2.1.1. Usability.

The overall usability for Xilinx ISE was measured at 57 hours. The usability for project preparation and tool familiarization took almost 35 hours. Much of this time was spent becoming familiar with the Xilinx ISE software, including installing the Spartan-3E development board. Preparation time was less since general knowledge about FPGAs and VHDL programming was gained in the Quartus II evaluation. As with the Quartus II evaluation, time was spent consulting many different sources to make the Hello World program work. The manufacturer provides many manuals for the system, but they assume a level of sophistication that not all novice users may possess. It is the author’s opinion that time could have been saved if a basic, entry-level tutorial specifically for this board was available in one document. Since the authors were using a program written by someone else, no time was spent developing it. However, approximately 8 hours were spent in the generation of the code itself. The time spent

measuring the four tasks took about 2 hours, which was done by logging all hours spent. Finally, the postmortem, where the data were collected and analyzed, took 12 hours.

#### F.5.2.1.2 Functionality.

In terms of functionality, the tester rated the following: tutorial—3, user manuals—4, readability—4.5, and flexibility—4. As noted before, it is the author's opinion that time could have been saved if a basic, entry-level tutorial, specifically for this board, was available in one document. As far as the user manuals themselves, they are high quality, providing detailed information in a logical and useful manner. The model development and code generation functionality was rated 3. Functionality seemed to be slightly higher because information seemed more readily available.

#### F.5.2.1.3 Efficiency.

In terms of efficiency, the code generated 310 bytes of memory.

#### F.5.2.2 Up-Down Counter.

The next step was to perform the four tasks again using Xilinx ISE to develop a simple up-down counter in VHDL code and run it on the Spartan-3E board (see section F.9). The VHDL program is identical to the one used for the Quartus II evaluation.

The following sections discuss the same four tasks as those performed in the Hello World program and the results of measuring the criteria for the up-down counter.

##### F.5.2.2.1 Usability.

The overall usability for Xilinx ISE was measured at 32 hours. The usability for project preparation and tool familiarization took almost 8 hours. As with the Hello World program, preparation time was less in the Xilinx ISE evaluation because general knowledge about FPGAs and VHDL programming was gained in the Quartus II evaluation. Therefore, usability, as measured in hours, is lower for the Xilinx ISE evaluation than for the Quartus II. About 8 hours were spent learning new features of the Xilinx ISE and the Spartan-3E development board that were not explored in the Hello World program. Since identical programs were used in the evaluation of both software tools, negligible time was spent designing and writing the VHDL code, making corrections to the code due to compile errors, and getting the up-down counter to run in the Spartan-3E board.

##### F.5.2.2.2 Functionality.

In terms of functionality, the tester rated the following: tutorial—3, user manuals—4, readability—4.5, and flexibility—4. Since the same documentation was used, the measured functionality was the same. The model development and code generation functionality was rated 3.5. The model development and code generation functionality was rated higher for the up-down

counter because of the experience gained in the Hello World program and the user's confidence in using Xilinx ISE.

#### F.5.2.2.3 Efficiency.

In terms of efficiency, the code generated 282 bytes of memory.

#### F.5.3 LabVIEW 8.5.

The LabVIEW 8.5 FPGA Module allows the user to program an FPGA with a LabVIEW block diagram. The module uses code-generation techniques to synthesize the graphical development environment to FPGA hardware. The designer uses graphical programming to create a highly optimized gate array implementation of analog or digital control logic. Normal LabVIEW programming techniques are used to develop FPGA applications, although when targeting FPGA hardware such as CompactRIO™, the LabVIEW programming palette is simplified to contain only the functions that are designed to work on FPGAs, the primary programming difference compared to traditional LabVIEW is that FPGA devices use integer math rather than floating-point math.

The LabVIEW FPGA Module compiles a LabVIEW application to FPGA hardware using an automatic multistep process (see figure F-8). First, graphical code is translated into text-based VHDL code. Next, the Xilinx ISE compiler is invoked, and the VHDL code is optimized, reduced, and synthesized into a hardware circuit realization of the user's LabVIEW design. Timing constraints are applied to the design to achieve an efficient use of FPGA resources. Optimization is performed during the FPGA compilation process to create an optimal implementation of the LabVIEW application and reduce the digital logic size. Next, the design is synthesized into an optimized silicon implementation, providing parallel processing capabilities. The final result is a bit stream file that contains the gate array configuration information. When the application is run, the bit stream is loaded into the FPGA chip and used to reconfigure the gate array logic [F-9].

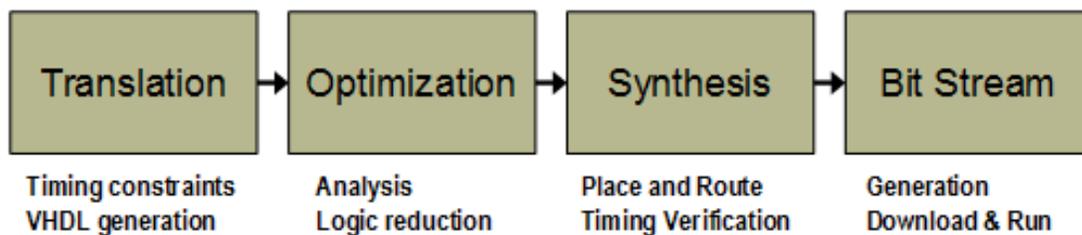


Figure F-8. LabVIEW FPGA Compilation Process

#### F.5.3.1 Hello World.

Due to the graphical nature of the LabVIEW environment, the approach used for the Hello World and up-down counter programs was different. A LabVIEW VI was created to implement the

Hello World program. To use the Hello World VHDL program in an FPGA VI, an HDL Interface Node was used rather than rewriting the code in LabVIEW. All the parameters and the VHDL code are entered in the HDL Interface Node Properties dialog box (see figure F-9).

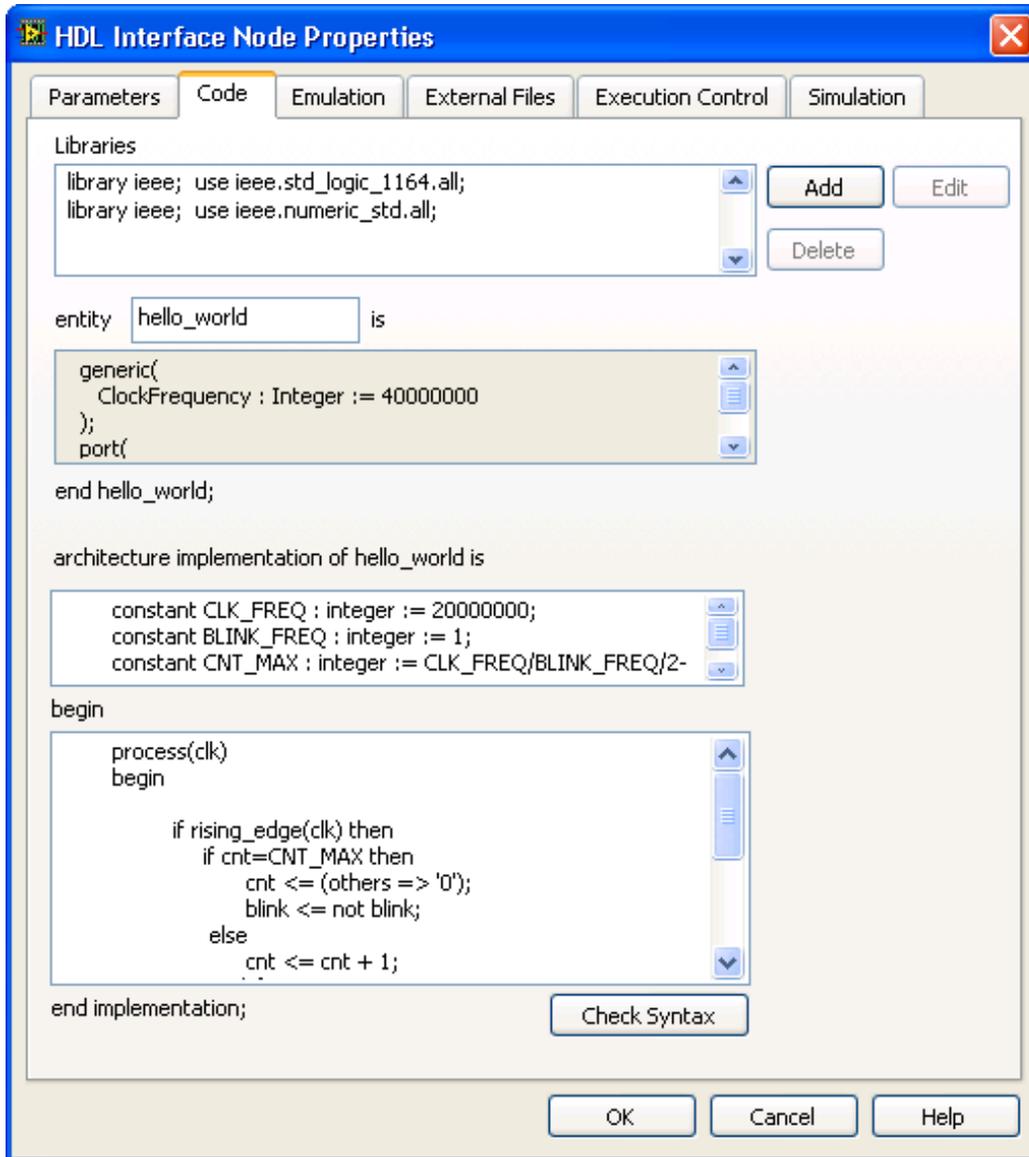


Figure F-9. HDL Interface Node Properties Dialog Box

The parameters become terminals on the HDL Interface Node. The parameters are then wired to any VI or function on the block diagram. For Hello World, it was necessary to add “led” in the Parameters tab (see figure F-10). In the “begin” section of the code tab, the statement “enable\_out <= enable\_in;” needs to be added. The HDL Interface Node uses an enable chain to follow the LabVIEW data flow model. The enable chain is the collection of signals and an associated protocol for controlling the HDL component’s input and output data flows. The HDL



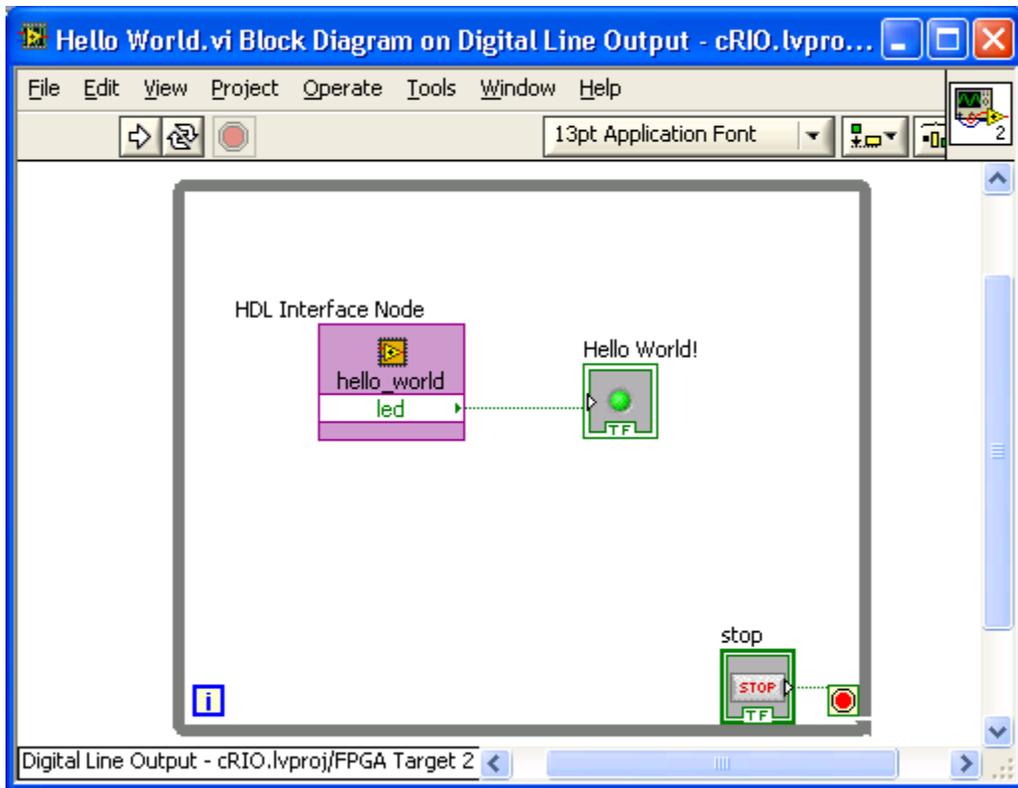


Figure F-11. Hello World VI Block Diagram



Figure F-12. Hello World VI Front Panel

The results of measuring the criteria are discussed in the following sections for the Hello World program.

#### F.5.3.1.1 Usability.

The overall usability for the LabVIEW program was measured at 35 hours. The usability for project preparation and tool familiarization took almost 20 hours. Much of this time was spent becoming familiar with the LabVIEW HDL Interface Node. Preparation time was less in the LabVIEW evaluation because general knowledge about FPGAs and VHDL programming was gained in the Quartus II evaluation. Also, the author had some previous LabVIEW experience. As in the Quartus II evaluation, time was spent consulting different sources to make the Hello World program work. The new feature requiring the most time was the HDL Interface Node. All information needed to learn how to make VHDL code run in the LabVIEW software was gathered from National Instrument websites—no printed tutorials were found. As before, it is the author's opinion that time could have been saved if a basic, entry-level tutorial specifically for this board was available in one document. Approximately 5 hours were spent in the generation of the code itself. The time spent measuring the four tasks took about 2 hours, which was done by logging all hours spent. Finally, the postmortem, where the data were collected and analyzed, took 8 hours.

#### F.5.3.1.2 Functionality.

In terms of functionality, the tester rated the following: tutorial—2, user manuals—2.5, readability—4.5, and flexibility—4. As noted before, it is the author's opinion that time could have been saved if a basic, entry-level tutorial (specifically for this board) was available in one document or on one website. The websites are high quality, providing detailed information in a logical and useful manner; however, information had to be found at several different websites to make Hello World work. The disjointed nature of gathering information from multiple websites hindered learning how to use the LabVIEW HDL Interface Node. The model development and code generation functionality was rated 2.5. Functionality seemed to be slightly lower because information was not readily available.

#### F.5.3.1.3 Efficiency.

In terms of efficiency, the code generated 941 KB of memory.

#### F.5.3.2 Up-Down Counter.

The next step was to perform the four tasks again, this time using LabVIEW to develop a simple up-down counter. Again, the VHDL program used for the previous evaluations was modified for use in LabVIEW. Using the HDL Interface Node Properties tab, the following parameters were entered, as shown in figure F-13.



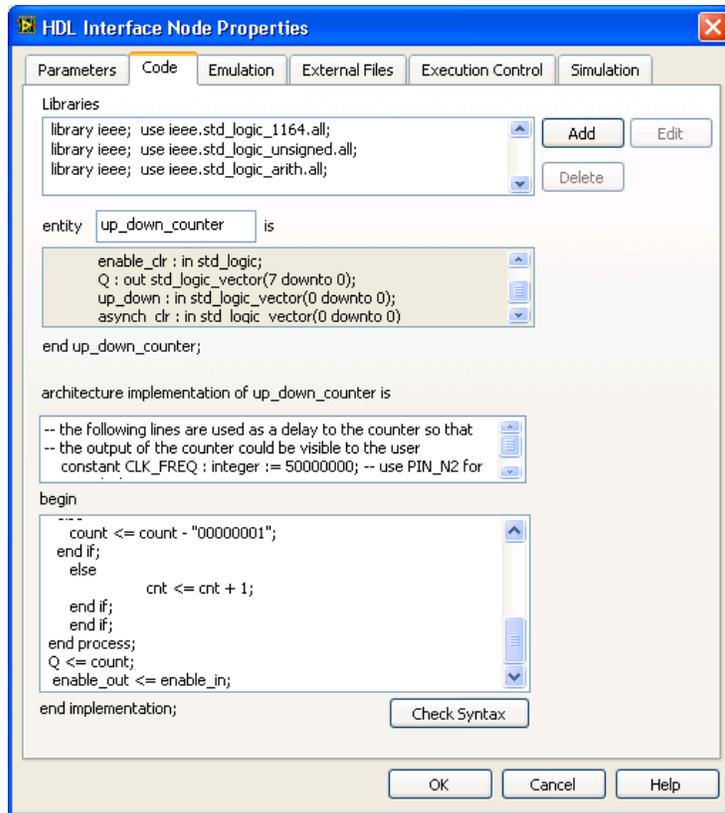


Figure F-14. Up-Down Counter HDL Interface Node Code Tab

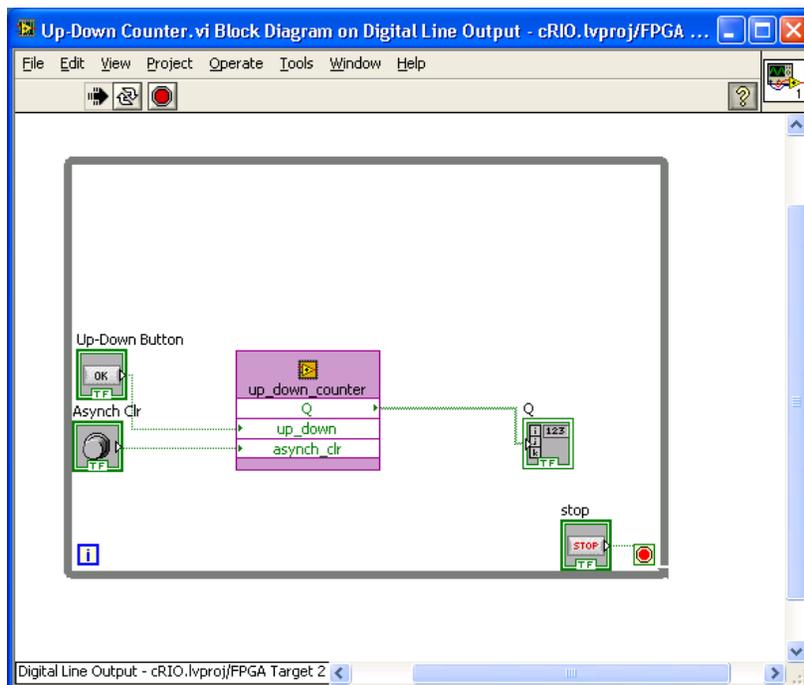


Figure F-15. Up-Down Counter Block Diagram

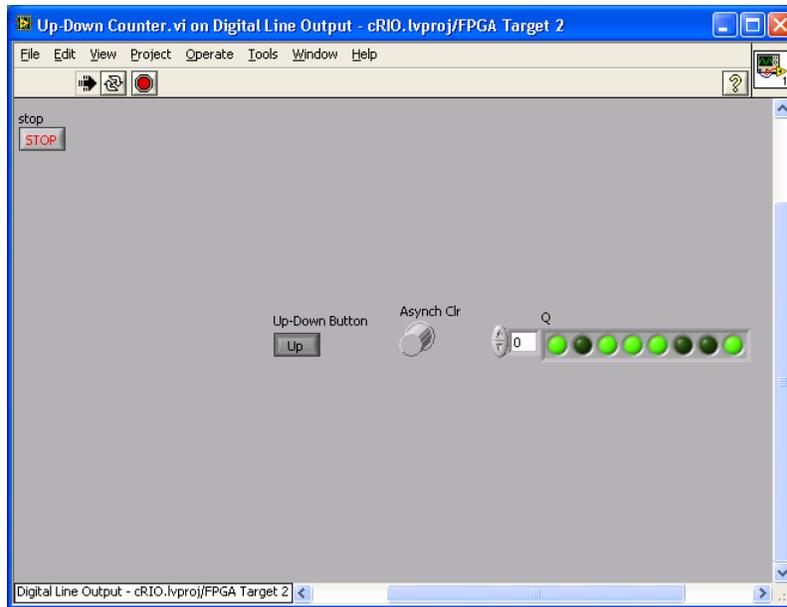


Figure F-16. Up-Down Counter Front Panel

The following sections discuss the four tasks performed in the Hello World program and the results of measuring the criteria for the up-down counter.

#### F.5.3.2.1 USABILITY.

The overall usability for LabVIEW was measured at 60 hours. The usability for project preparation and tool familiarization took almost 5 hours. As with the Hello World program, there was less preparation time because general knowledge about FPGAs and VHDL programming was gained in the previous evaluations. The bulk of the time was spent modifying the Up-Down Counter VHDL code so it could be used in the HDL Interface Node. As shown in figure F-13, three parameters were entered, Q, up\_down, and asynch\_clr. In section F.9, Q is an eight-element std\_logic\_vector, and up\_down and asynch\_clr are std\_logic data types. However, in figure F-14, note that the HDL Interface Node codes are up\_down and “asynch\_clr” as “std\_logic\_vector(0 downto 0)”. After searching various websites, it was determined that statements in the original VHDL code had to be modified to be compiled. First, “if (asynch\_clr='0') then” had to be changed to “if (asynch\_clr=“0”) then” and “if (up\_down='1') then” had to be changed to “if (up\_down=“1”) then.” This is because double quotes (“ ”) are used for vectors and single quotes (‘ ’) are used for the one-digit scalar std\_logic. To date, the author has not uncovered the reason that one-digit scalars are coded as std\_logic\_vector(0 downto 0) in LabVIEW. These are simple changes, but the opinion was that too much time was spent searching various websites trying to resolve this problem when an explanation from NI as to what needed to be modified in standard VHDL code should have been given. NI should list all changes that developers need to make to their existing VHDL code to have that code run properly in the HDL Interface Node.

The time spent measuring the four tasks took about 2 hours, which was done by logging all hours spent. Finally, the postmortem, where the data were collected and analyzed, took 8 hours.

#### F.5.3.2.2 Functionality.

In terms of functionality, the tester rated the following: tutorial—2, user manuals—2.5, readability—4.5, and flexibility—4. Since the same documentation is used, the measured functionality was the same. Model development and code generation functionality was rated 3.5. Model development and code generation functionality was rated higher for the up-down counter because of the experience gained in the Hello World program and the user’s confidence in using Xilinx ISE.

#### F.5.3.2.3 Efficiency.

In terms of efficiency, the code generated 941 KB of memory.

### F.6. DISCUSSION OF RESULTS AND COMPARISON.

Four experiments with the Quartus II and Xilinx tools were conducted for the Hello World and up-down counter, as explained in section F.4. The criteria and methodology, as discussed in section F.3, were applied.

#### F.6.1 FUNCTIONALITY.

Figure F-17 shows the cumulative results of measuring functionality of all the tools: the Altera Quartus, Xilinx ISE, and NI LabVIEW FPGA module.

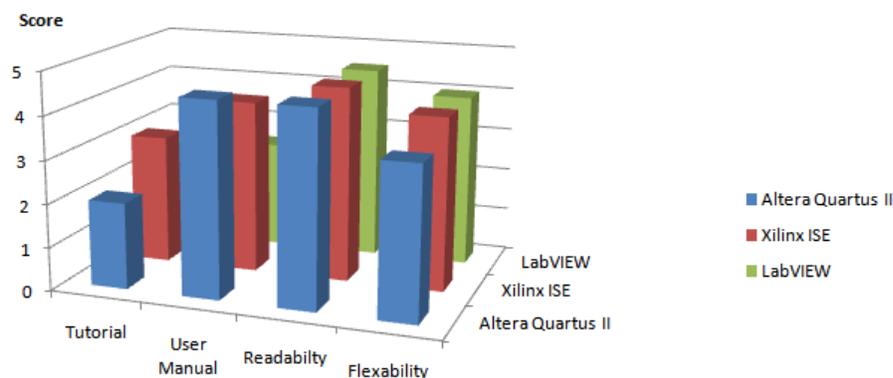


Figure F-17. Comparison of Functionality Measured for All Tools

It appears that all the tools have poor tutorials, and there is a difference between the tutorials for all the systems. The opinion was that Xilinx provided better tutorials than Altera. The documentation provided by Altera was comprehensive and complete but, for the novice user, there seemed to be no clear starting point at which to begin. As an example, time had to be spent searching for the necessary pin assignments in order to compile the VHDL code for the Quartus

II board. Less time was required for the Xilinx board since the pin assignments were printed directly on the board. Thus, time was saved learning how to compile VHDL code for the Xilinx ISE regardless of general knowledge and experience. LabVIEW was judged the least favorable in terms of functionality. This was mainly due to the issues regarding the HDL Interface Node, as described in section F.5.3.1.

### F.6.2 USABILITY.

Figure F-18 shows the comparison of usability between Altera Quartus II, Xilinx ISE, and NI LabVIEW for the Hello World program.

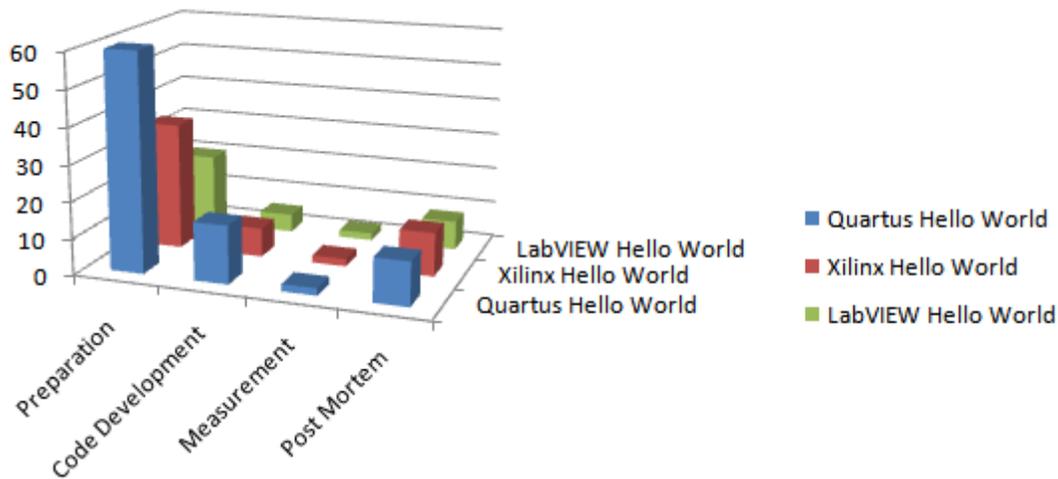


Figure F-18. Comparison of Usability Measured for All Tools

Measurement and postmortem were essentially the same, which is consistent with the fact that these activities are conducted regardless of the platform used. The big difference is in preparation. More time was spent in preparation with Quartus II than with the Xilinx ISE. The least amount of preparation time was spent with LabVIEW. The fact that the author, being new to development tools for FPGAs, began this research with Quartus II would account for some of this difference. If this study began with the Xilinx ISE, these numbers would obviously change, and the usability of the Quartus II would be higher (i.e., fewer hours). Also, since the author had more experience and knowledge of FPGAs and their development tools when measuring the usability of the Xilinx ISE compared to the Quartus II, this would show how experience is a factor in how usability is perceived. But functionality also played a role.

Figure F-19 compares the usability of the up-down counter for all tools. The time preparation for Quartus II, Xilinx ISE, and LabVIEW was essentially the same. The difference here was during code development. Since the code for the up-down counter was written during the learning phase for the Quartus II software, more time was spent writing the VHDL code and debugging it. Again, LabVIEW was judged least usable due to the documentation reviewed.

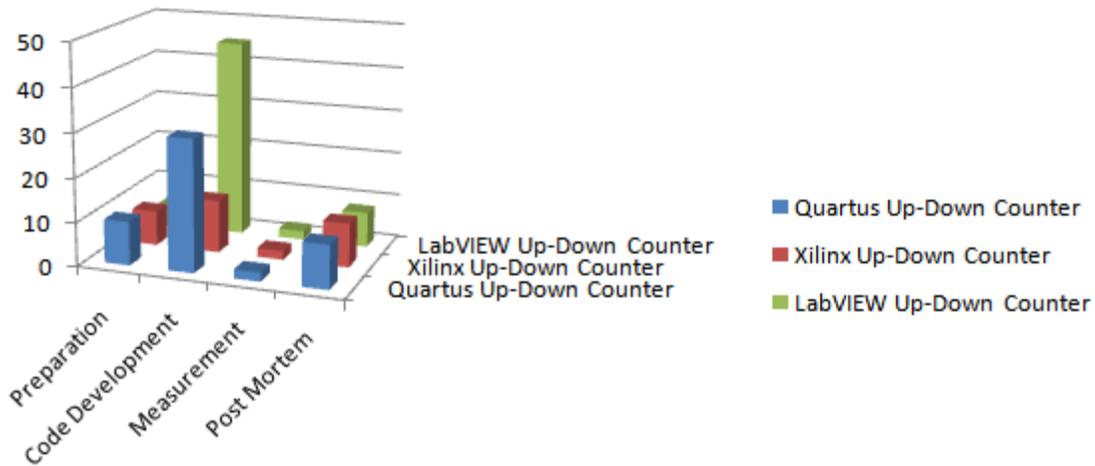


Figure F-19. Comparison of Usability Measured for All Tools

Figures F-20 through F-22 compare the usability between Hello World and the up-down counter for all tools. The time spent preparing Hello World with the tools is significantly higher than the of up-down counter, but that may not be significant because Hello World was the first program developed with each tool so the learning curve is included in preparation time. The code development time for up-down counter is slightly higher than for Hello World. This is because the up-down counter’s code is more complicated.

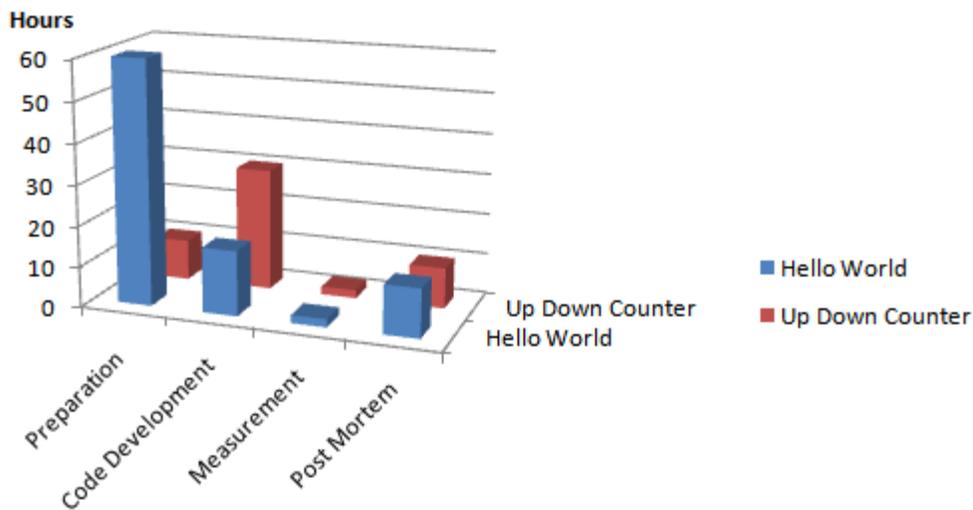


Figure F-20. Usability Measured for Quartus II (in hours)

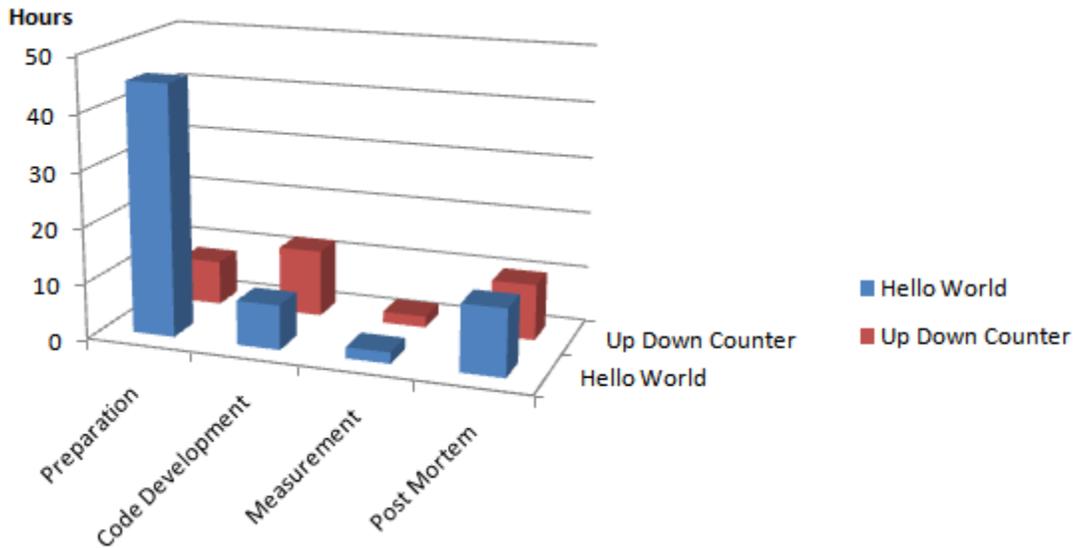


Figure F-21. Usability Measured for Xilinx ISE (in hours)

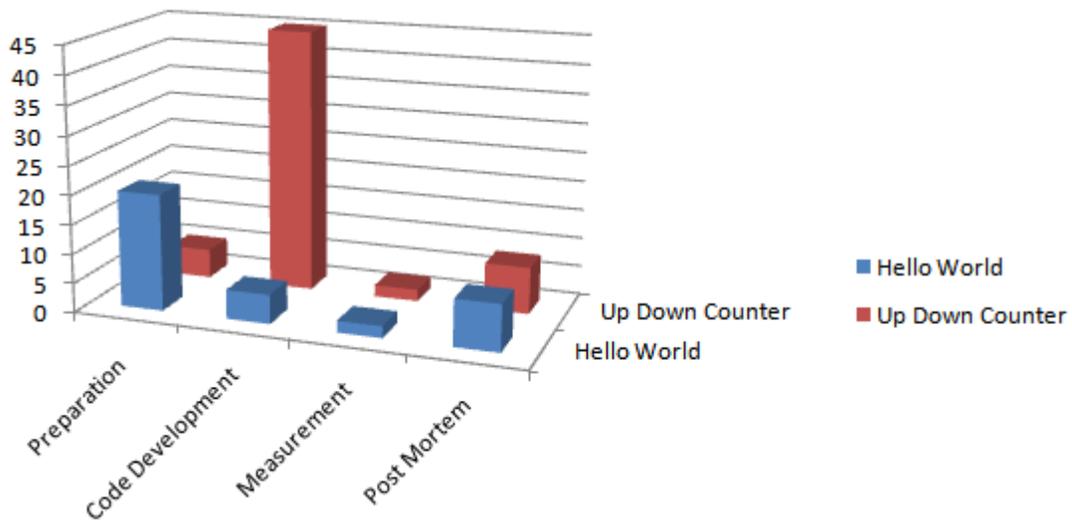


Figure F-22. Usability Measured for LabVIEW (in hours)

F.7 SUMMARY.

The objective of this research was to measure the usefulness of the Altera Quartus II Design Suite, Xilinx ISE Design Suite, and NI LabVIEW 8.5 FPGA module hardware design tools. There were several problems encountered as to what constitutes tool quality. Tool quality itself is a subjective quality, with the quality of the same tool being judged differently by different users. The problem of establishing the quality of a tool then becomes how to define what features constitute “quality” and how to measure them, while at the same time trying to ensure that personal bias does not play a dominant role. Users have different experiences and a different knowledge base that they bring. The user’s views on the quality of a particular tool are important

and must play a role. The challenge is to balance the user's opinions with quantitative data. If too much emphasis is placed on subjective opinion, then measuring the quality of hardware design tools becomes an exercise in product review. To have a meaningful measure of tool quality, it is important that a large sample of users with diverse experience be used.

At the beginning of this study, the authors had little experience using design tools for FPGAs. When considering the usability measurements, the fact that the authors gained knowledge and experience as the study progressed must be taken into consideration. The first tool to be studied was the Altera Quartus II. It was with this tool that the authors learned about FPGAs and VHDL. Therefore, it would not be fair to compare the usability of the Quartus II to the Xilinx ISE and the LabVIEW FPGA module, within the context of this study, since most of the preparation and code development was done for the Quartus II. For example, the time spent learning to code, in VHDL, the up-down counter was done during the Quartus II phase of this study. For the Xilinx ISE, the code used for the up-down counter was exactly the same. This is reflected in the usability measurements where more hours were spent for the Quartus II than the Xilinx ISE. The code had to be modified in order to work within the constraints of HDL Interface Node for LabVIEW FPGA module. The changes needed were not that difficult to make. The difficulty came as a result of the user having to search how to make those changes. For all the tools there seemed to be a direct correlation between usability and documentation. It is felt from these experiments that better, more accessible documentation of the tools by the manufacturers would lead to better usability ratings for these tools.

These tools are used to assist the designer to upload their design to an FPGA. They are very efficient in producing FPGA circuits that satisfy all the design rules of the target technology, such as hold times, maximum fan-out, and connection rules. They provide many ways that the designer can simulate his circuit to verify the design's behavior is logically correct at several levels of abstraction. But the design tools do not guarantee the logical correctness of the design itself. Designers usually have a limited amount of time to run simulations of their design to verify correctness. It is always possible that flaws remain in the design after testing. This is one of the reasons for the popularity of FPGAs. If bugs are found after a design is in the field, it is usually a simple matter of having a technician upload the corrected design to the FPGAs already deployed rather than replace an entire board.

This study was not meant to be the definitive study of tool quality of the Quartus II, Xilinx ISE, and LabVIEW FPGA hardware design tools. What was attempted was to provide a framework for evaluating the quality of hardware design tools. The term "quality" is abstract in its nature. This study attempted to quantify quality so it could be measured. To more fully measure quality, a larger group of developers with various degrees of knowledge and experience would be needed so a true understanding of the tool's usefulness would emerge.

## F.8 HELLO WORLD CODE.

```
--
-- hello_world.vhd
--
-- The 'Hello World' example for FPGA programming.
--
-- Author: Martin Schoeberl (martin@jopdesign.com)
--
-- 2006-08-04 created
--
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity hello_world is

port (
    clk : in std_logic;
    led : out std_logic
);
end hello_world;

architecture rtl of hello_world is

    constant CLK_FREQ : integer := 20000000;
    constant BLINK_FREQ : integer := 1;
    constant CNT_MAX : integer := CLK_FREQ/BLINK_FREQ/2-1;

    signal cnt : unsigned(24 downto 0);
    signal blink : std_logic;

begin

    process(clk)
    begin

        if rising_edge(clk) then
            if cnt=CNT_MAX then
                cnt <= (others => '0');
                blink <= not blink;
            else
                cnt <= cnt + 1;
            end if;
        end if;

    end process;

    led <= blink;

end rtl;
```

## F.9 UP DOWN COUNTER CODE.

```
-- up_down_counter.vhd
-- Joseph Voelmle
-- 12-22-2008 created
--
library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_arith.all ;
use ieee.std_logic_unsigned.all;

entity up_down_counter is

port (clk, up_down, asynch_clr: in std_logic;
      Q: out std_logic_vector(7 downto 0)
);
end up_down_counter;

architecture counter_behavior of up_down_counter is

-- the following lines are used as a delay to the counter so that
-- the output of the counter could be visible to the user
constant CLK_FREQ : integer := 50000000; -- use PIN_N2 for 50MHz clock
constant BLINK_FREQ : integer := 5;
constant CNT_MAX : integer := CLK_FREQ/BLINK_FREQ;

signal count: std_logic_vector(7 downto 0);
signal cnt : unsigned(24 downto 0);

begin -- count is an internal signal to this process
  process(clk, asynch_clr) -- sensitivity list
  begin
    if (asynch_clr='0') then -- asynch_clr is Pushbutton[0]
      count <= "00000000";
    elsif (rising_edge(clk)) then
      -- after so many clocks, increment/decrement count
      if cnt=CNT_MAX then
        cnt <= (others => '0');
        -- up_down is Toggle Switch[0], up position count up,
        -- down count down
        if (up_down='1') then
          count <= count + "00000001";
        else
          count <= count - "00000001";
        end if;
      else
        cnt <= cnt + 1;
      end if;
    end if;
  end process;
  Q <= count;
end architecture counter_behavior;
```

## F.10 REFERENCES.

- F-1. Kornecki, A. and Zalewski, J., "Experimental Evaluation of Software Development Tools for Safety-Critical Real-Time Systems," *Innovations in System Software Engineering*, Vol. 1, 2005, pp. 176-188.
- F-2. Dahll, D. and Gran, B.A., "The Use of Bayesian Belief Nets in Safety Assessment of Software Based Systems," *Int. J. General Systems*, Vol. 29 (2), pp 205-229.
- F-3. Dahll, D., "Safety Assessment of Programmable Systems Containing COTS Components," Proceedings of the ENCRESS Workshop "Using COTS in Component-Based Systems for Dependable Applications," Naples, May 23, 2000.
- F-4. Gran, B.A., Dahll, D., Eisenger, S., Lund, E.J., Nordtrøm, J.G., Strocka, P., Ystanes, B.J., "Estimating Dependability of Programmable Systems Using BBNs, Proceeding of SAFECOMP2000," *Intern. Conf. on Safety, Reliability and Security*, Springer-Verlag, Berlin, 2000, pp. 309-320.
- F-5. Introduction to the Quartus® II Software, Version 8.0, Altera Corp. San Jose, California, [www.altera.com/literature/manual/intro\\_to\\_quartus2.pdf](http://www.altera.com/literature/manual/intro_to_quartus2.pdf).
- F-6. Schoeberl, M., The FPGA Hello World Example, August 4, 2006, [www.jopdesign.com/cyclone/hello\\_world.pdf](http://www.jopdesign.com/cyclone/hello_world.pdf).
- F-7. IEEE Std 1061-1998, IEEE Standard for a Software Quality Metrics Methodology, The Institute of Electrical and Electronics Engineers, Inc., New York, December 8, 1998.
- F-8. ISE 10.1 Quick Start Tutorial, Xilinx, Inc., San Jose, California, [www.xilinx.com/itp/xilinx10/books/docs/qst/qst.pdf](http://www.xilinx.com/itp/xilinx10/books/docs/qst/qst.pdf).
- F-9. FPGA-Based Control: Millions of Transistors at Your Command, National Instruments Corp, Austin, Texas, <http://zone.ni.com/devzone/cda/tut/p/id/3357>.