Issues in Tool Qualification for Safety-Critical Hardware: What Formal Approaches Can & Cannot Do

Brian Butka[1], Janusz Zalewski[2], Andrew J. Kornecki[3],

[1]Electrical Engineering, Embry Riddle Aeronautical Univ., Daytona Beach, FL 32114 USA
[2]Computer Science, Florida Gulf Coast University, Fort Meyers, FL 33965 USA
[3]Computer&Software Engineering, Embry-Riddle Aeronautical Univ., Daytona Beach, FL 32114 USA

[1]butkab@erau.edu, [2]zalewski@fgcu.edu, [3]kornecka@erau.edu

Abstract. Technology has improved to the point that system designers have the ability to trade-off implementing complex functions in either hardware or software. However, clear distinctions exist in the design tools. This paper examines what is unique to hardware design, areas where formal methods can be applied to advantage in hardware design and how errors can exist in the hardware even if formal methods are used to prove the design is correct.

Keywords: Tool Qualification, HDL, PLD, Hardware Design, Safety-Critical Systems, Formal Methods

1 Introduction

Safety-critical applications, such as a modern aircraft use not only increasing numbers of microprocessors and microcontrollers but also dedicated hardware to process the growing amounts of data needed to control the flight and related systems, and monitor their status. Rapid progress of digital technology in the last 25 years can be shown on example from Airbus industries: the increase of number of digital units from 70 to 300, number of transistors from 10^5 to 10^8, and number of gates per chip from ten to 600 thousand [1]. Recent proliferation of custom micro-coded components changed the market and the ways how the industry operates. These complex programmable electronic components are not only programmed using conventional programming languages but their logic designs are also developed by writing code in a Hardware Description Language (HDL) such as VHDL, Verilog, or SystemC. The two distinctive categories of modern electronic components are programmable logic devices (PLD) and application specific integrated circuits (ASIC). PLD is purchased as standard electronic parts and then altered (or programmed) to perform specific function. ASIC is developed using an expensive process of fabrication with the design embedded in the layers of silicon. Once manufactured, ASIC cannot be re-

programmed and thus it is not a PLD – although the original program for ASIC is developed in HDL, in the same manner as for PLD. The primary and most popular PLD components type are field programmable gate arrays (FPGA), often treated as a separate category. The scope of this research has been limited to tools supporting development of FPGA that have been used, or have a potential to be used, in airborne applications.

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 programmable logic device is literally software. Software tools are used to simulate the logic, synthesize the circuit, and create the placement and routing for the electronic elements and their connections in preparation for the final implementation, i.e., programming the logic devices, which used to be conventionally called "burning into the logic." The development of hardware relies significantly on the quality of tools, which translate the software artifacts from one form to another. Both software and hardware use extensively very complex tools, i.e., integrated programming environments taking the project from its conceptual stage into the final product. Thus, the quality of such tools is essential for the proper operation of respective products in the real-life environment, especially in safety-critical systems, where computer systems may cause unintended harm to human life or property.

The objective of this work is to analyze the use of software tools in hardware development for safety-critical systems, from the perspective of potential application of formal approaches to improve product quality. The rest of the paper is structured as follows. Section 2 sets the stage for the analysis, providing an overview of a design flow for PLD components, with emphasis on design verification. Section 3 outlines the potential impact of tool quality on product safety, and Section 4 discusses specific hardware issues that can still remain unresolved after formal verification of the design.

2 PLD Design Flow and Formal Approaches

A generic PLD design flow is shown in figure 1. The PLD design process begins by describing the hardware functionality in an HDL. The HDL code can then be simulated to verify correct function at the behavioral level. The synthesis process converts the high-level HDL code into a netlist of interconnected logic functions. The place and route process fits the netlist to the vendor specific hardware architecture of the PLD. The synthesis and place and route processes provide opportunities for errors to be introduced to a logically correct HDL description of a design. In the rest of this section, we review the formal approaches addressing some of these problems from the tools perspective, as well as outline an engineering view based on practical experience with similar issues.

2.1 Review of the Use of Formal Approaches in Hardware Design Tools

A thorough review of literature for the last decade, or so, reveals a number of attempts to formalize reasoning about hardware design, for example [2], with very few of them related to tools. A handful of selected papers are mentioned below, in chronological order. For the purpose of this discussion, we follow the definition of a formal method as given by the NASA Langley Formal Methods Group:

"Formal Methods" refers to mathematically rigorous techniques and tools for the specification, design and verification of software and hardware systems. The phrase "mathematically rigorous" means that the specifications used in formal methods are well-formed statements in a mathematical logic and that the formal verifications are rigorous deductions in that logic (i.e. each step follows from a rule of inference and hence can be checked by a mechanical process.) (http://shemesh.larc.nasa.gov/fm/fm-what.html)

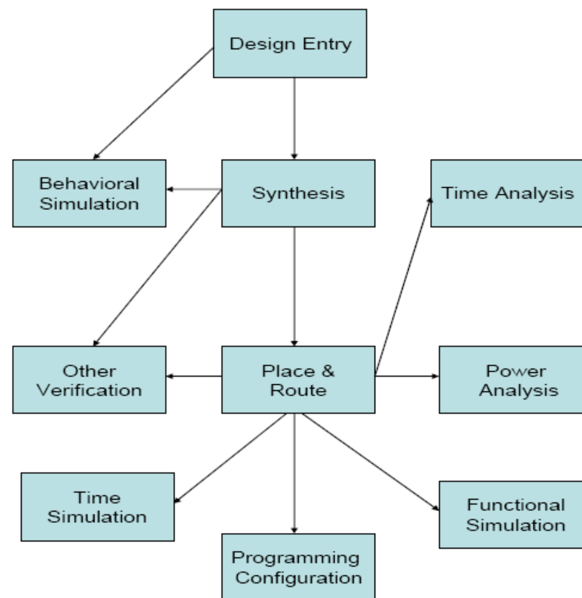

Fig. 1. A generic PLD design flow

In an overview paper [3] Kern and Greenstreet identify two main aspects of the application of formal methods in a hardware design process: (a) the formal framework used to specify desired properties of a design, and (b) the verification techniques and tools used to reason about the relationship between a specification and a corresponding implementation. They survey a variety of frameworks and techniques proposed in the literature as applied to actual designs. The specification frameworks include temporal logic, predicate logic, regular languages, abstraction and refinement. The verification techniques include model checking, automata-theoretic techniques, automated theorem proving, and approaches that integrate the above methods. The paper presents 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.

Only relatively recently authors of papers on formal methods began considering tools supporting these approaches. A handful of related papers are discussed below. Turner and He [4] investigate specification, verification and test generation for synchronous and asynchronous circuits. Their approach is based on temporal ordering specification using Digital Logic In LOTOS (DILL). The paper defines relations for strong conformance to verify a design specification against a high-level specification, and describes tools for automated testing and verification of conformance between an implementation and its specification.

Aljer and Devienne [5] consider the use of a formal specification language as the foundation of real validation process. They propose architecture based upon stepwise refinement of a formal model to achieve controllable implementation. Partitioning, fault tolerance, and system management are seen as particular cases of refinement in order to conceptualize systems correct by proven construction. The methodology based on the refinement paradigm is described. To prove this approach, the B-HDL tool based on a combination of VHDL and B method formal language has been developed.

Nehme and Lundqvist [6] describe a framework combining software tools for application verification and hardware platforms for execution and real-time monitoring. The tool translates safety critical VHDL code into a formal representation in a form of finite state machine (FSM) model. Formal techniques can then be applied on FSM representation to verify properties such as liveness and deadlock and to validate that the timing constraints of the original system are met. Three aspects of the tool implementation are discussed: transformation of source code into an intermediate representation, verification of real-time properties, and some tool-related implementation issues.

Dajani-Brown et al. [7] focus on the use of SCADE (Safety Critical Application Development Environment) 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 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. 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) approaches used for modeling the voter's environment; and 2) the strengths and shortcomings of such approaches when applied to the discussed problem.

Hilton in his thesis [8] 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 as a semantic basis for specifying and proving properties of programs executing on PLD, and extends the use of SPARK Ada to cover the interface between software and programmable logic. The author claims that the demonstrated methods are not only feasible but also scale up to realistic system sizes, allowing development of such safety-critical software-hardware systems to the levels required by current system safety standards.

Finally, with the emergence of the FAA endorsed document DO-254 “Design Assurance Guidance for Airborne Electronic Hardware” [9], more papers began to appear that discuss not only tool support for formal approaches, but also compliance with the DO-254 standard. This is where initial discussions of product or process certification and tool qualification begin to take place.

Dallacherie et al. [10] look at a static formal approach that may be used, in combination with requirements traceability features, in the design and verification of hardware controllers to support such protocols as ARINC 429, ARINC 629, MIL-STD-1553B, etc., with respect to compliance with DO-254. The paper describes the application of a formal tool in the design and verification of airborne electronic hardware developed in a DO-254 context. imPROVE-HDL tool 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 the need to create testbenches. Another tool, Reqtify, is used to track the requirements and produce coverage reports throughout the verification process. The authors claim that 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.

Karlsson and Forsberg [11] discuss the additional strategies identified in RTCA DO-254 for the highest levels of design assurance (A and B). In particular, the use of formal property specification language (PSL) in combination with dynamic (simulation) and static (formal) verification methods for programmed logic devices are addressed. Using these methods, a design assurance strategy for complex programmable airborne electronics compliant with the guidelines of RTCA DO-254 is suggested. The proposed strategy is a semi-formal solution, a hybrid of static and dynamic assertion based verification. 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.

As shown above, formal approaches have some demonstrated successes in hardware design; however, the essence of formal methods is that they require a perfect model of the physical system. Thus, due to the complexity of actual systems, formal approaches can be only used in parts of the design process. Typically, formal methods are used early in the development life cycle substituting formal abstraction for a complete physical model. Subsequent refinement is then used to map forward requirements to the later stages of the life cycle.

2.2 Engineering Approach to PLD Design Verification

Simulation, which requires the generation of appropriate test vectors, 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 has been considered adequate to verify that the design specified in HDL Register Transfer Language (RTL) performs the intended function at the behavioral level. Verification of million-gate designs at the gate level requires that transitions on every gate be tracked, resulting in runtime of weeks for substantial million-gate designs.

Since an RTL design can be implemented in a variety of ways on 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.

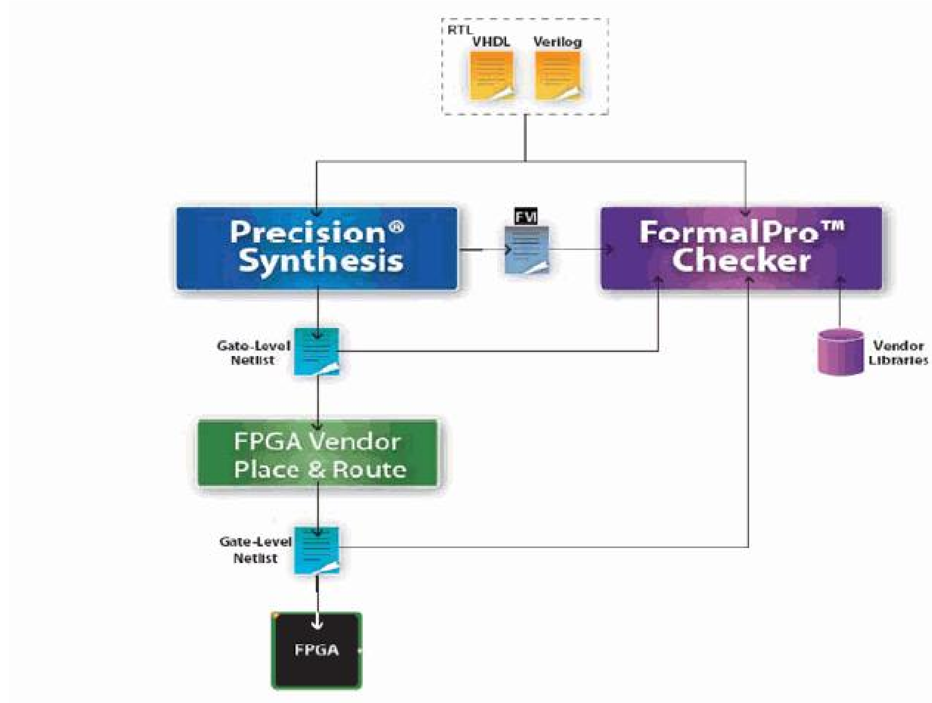

Fig. 2. Verification with Formal Equivalence Checking [12]

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 solution to this problem could be the utilization of formal methods. 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, correct, and does not change the semantic properties of the original input artifact.

One such approach is an equivalence checker, which uses static verification techniques to prove that the RTL and gate-level representations of digital design are an exact functional match. Full verification at the gate-level simulation for modern million-gate designs is infeasible. A formal checker (figure 2 [12]) uses a formal verification interface file (FVI) 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, it can be assumed the final design is consistent with the original design intent.

Assuming that the original RTL representation (in VHDL or Verilog) verified by extensive simulations is functionally correct, equivalence checking is an acceptable solution, since it ensures that transformations throughout the design flow comply with the original

functionality. However, equivalence checking does not replace timing analysis. Static and dynamic timing analysis tools should still be used to confirm gate-level timing.

Despite the obvious advantages of the formal equivalence checking approach, there are limitations. Formal tools appear to provide the ultimate assurance of design correctness. At the end of a run, the program provides counter-examples for each specified property specified by developer which were found not valid. Every property is 100% covered. But the Achilles heel of the process is determining how completely the set of properties covers the design intent. This requires "human-in-the-loop" - the skill of experienced designers.

There are numerous safety issues for designers to consider during the synthesis and place and route processes of a hardware design. The related potential errors are often caused by unexpected optimizations occurring during the tool-driven synthesis process. Because these errors occur while translating from the HDL description to the hardware implementation, the resulting design may be faulty even though the HDL implementation has been proven to be correct. In the next section, we take a closer look at the issues relevant to tool use in the design synthesis process.

3 Safety Issues

The case studies have been developed based on the expressed concerns of the airborne systems developers and certifying authorities with the reference to tools used for FPGA development under the FAA mandated RTCA DO-254 guidelines [9]. Largely, these concerns are relevant to development of all safety-critical and real-time systems. The approach was that the tools will be used in worst-case least-likely use scenarios, to test the bounds of the tools' capability. The black box design entered into the tool shall have a one-to-one mapping trace to the black box operation that is finally implemented. To facilitate design independence, case studies are very simple cases exploring specific attributes of a tool. This method has been selected over a large elaborate design to avoid unnecessary issues related to flaw in the design itself. The case studies address timing constraints, power integrity, and undefined input/output states. Additionally, the research explored issues of differences between behavioral simulation and implemented circuit behavior as well as tool awareness of circuit implementation on a faulty hardware.

3.1 Background

The design synthesis process is highly customizable and varies significantly from tool vendor to vendor. The variety of options and configurations make it difficult for an inexperienced designer to know exactly what the default synthesis settings are. Certain functions of synthesis, such as VHDL interpretation, are standardized by IEEE [13]. However, non-standard optimization techniques constitute the trade secret and are considered a competitive advantage of a given vendor. The tool user or designer often does not know the details of synthesis algorithms and thus is not aware of how the tool works. The magnitude of change of the intended design in the synthesis process and thus the impact on the final design may not be precisely known. The impact of the change depends upon the intricacies of the actual logic design, the selected tool used for synthesis, and the tool's

current settings. Regrettably, synthesis is not a standardized process; each vendor's tool is different. The differences are only known by a comparison of input versus output of different tools. Due to obvious reasons dealing with intellectual property 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 routable circuit from the HDL code that meets the performance goals is accomplished by merging logical synthesis and physical implementation 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 intellectual property of specific tool vendors. The general underlying background is well known, but the specifics of algorithms are not.

Safety is obviously an overall system property depending on behavior of hardware circuit as well as software that are developed to monitor and control the system. The issues discussed in the following sections have clearly impact of the ultimate safety of the system. The confusion between what the designer think the circuit (or the algorithm) will do versus what the actual physical circuit (or the running program) does is the main reason for potential safety violation.

3.2 Synthesis Issue #1 - Getting less than expected

The default configuration for almost all FPGA design tools is that all of the compiler and synthesis optimizations are enabled. This can lead to unexpected implementations. For instance, to reduce a design's sensitivity to single event upsets (SEU) errors a designer may write HDL code to specify a triple redundant module as shown in figure 3(A). However, the synthesis tool may determine that most of the hardware is redundant and implement the system as shown in figure 3(B). The independent multipliers were identified as redundant and optimized away during synthesis.

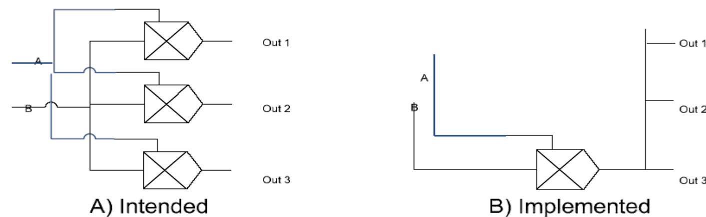

Fig. 3. Triple Redundant Module with Three Multipliers

3.3 Synthesis Issue #2 - Getting more than expected

In order 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,

particularly in systems where part of the circuit is monitoring the performance of another circuit. In designs that are intended to be tolerant of SEU, it is common to monitor that a critical flip-flop's outputs are logical opposites of each other under all conditions. Consider the circuit of figure 4(a) where the Output and the Monitor are always logically opposite. A logically equivalent implementation that could be generated by the synthesizer to meet timing constraints is presented in 4(b). In such a solution, a single event upset of the top flip-flop will not affect the monitor output. However, the resulting circuit does not guarantee that Output and Monitor are logical opposites, which defeats the purpose of the monitor output.

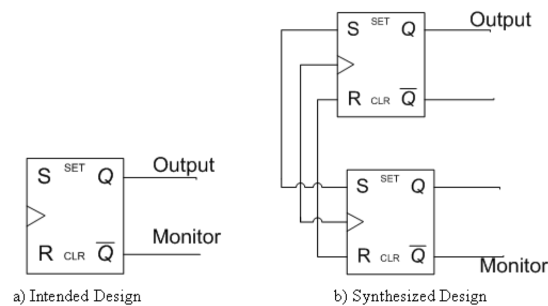

Fig. 4. Flip-Flop Replication

3.4 Synthesis Issue #3 - Hardware that is non-functional in normal operation

The triple redundant module and metastability examples take place when design optimizations are applied during synthesis. Typically, the software tool would not generate explicit warnings that the optimization had been used. The additional circuitry should not be active during the normal operation. Therefore this error may not be detectable during hardware validation since validation is performed using working hardware. The only methods of verifying correct operation of this circuitry is via simulation of the HDL. Since we cannot independently verify the operation of this circuitry in the hardware, the gate-level implementation must be verified. Formal methods offer the most appealing solution to this problem.

Since the above problems are caused by the synthesizer performing optimizations, we could turn off all synthesizer optimizations. Occasionally it can be a feasible solution. However, in most cases it is difficult to meet the timing and the chip area constraints without optimizations. An experienced designer would recognize such issues and configure the synthesizer optimizations appropriately. It is still difficult to assess if the designer has handled all possible areas of concern. There is evident need for a testing/design process where the experience of both the design and verification teams is considered in determining the level and rigor of verification that must be demonstrated.

4 Hardware Specific Issues

Even if a design has been formally verified by a tool of respective pedigree, it is still possible for hardware circuits to produce incorrect results. The hardware-related issues can be broadly classified as timing, signal, and power integrity issues.

4.1 Timing Issues

Perhaps the most difficult aspect in verifying the correctness of hardware is that even minor changes in the timing can produce major differences in the logical operation of a circuit. 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, since the simulation timing step size is typically much larger than the timing differences, the timing differences will not show up in the simulation output. Designers 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 and these effects are rarely (if ever) included in logic simulator models. 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.

4.1.1 Synchronous Design

To overcome the problem of not knowing when the data is valid, almost all hardware designs use a synchronous design with a clock. In this design style, the data is valid for some time before the clock edge (setup time) and for some time after the clock (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 devices 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 design 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, without notifying the designer. Speed differences between the clock and the data path may result in

the failures due to the data arriving too soon or too late. These failures are particularly sensitive to variations in temperature and voltage and often are concealed in simulation.

4.1.2 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. Correct operation of circuits crossing clock domain boundaries cannot be guaranteed by simulation because the timing between different clock domains can vary arbitrarily which would require an infinite number of simulations. Special design techniques are used to allow signals to cross between clock domains and designers must insert them where needed.

4.1.3 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. Modern designs use increasing number of clocks. 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 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 which varies depending on the exact timing of the inputs. This delay can only be analyzed statistically. We cannot prevent an error from happening; we can only bind its probability. Although most designers avoid asynchronous design, there are cases where such solution is required. An example would be a reset path that must operate, even if the synchronizing clock is not present.

4.2 Signal Issues

4.2.1 Combinational Feedback and Quasi-digital Circuits

PLDs provide the user with the ability to configure the device a nearly infinite number of ways. This flexibility can allow the designer to implement unexpected configurations. For instance, it is possible to configure an odd number of inverter 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 5). 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 oscillations. Most HDL simulators assume only digital logic and are unable to correctly simulate this simple analog 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, we must restrict the designer's ability to produce problematic configurations.

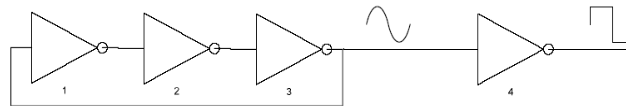

Fig. 5. Ring Oscillator

4.2.2 Undefined States and Constant Signals

FPGAs may contain large numbers of states which are defined as “don't care” for certain modes of operation. Many inputs and internal variables are often defined as constants. Different synthesis tools handle the “don't care” states and constants quite differently. This makes formal verification a very user intensive process requiring manual customization of the verification tool.

4.3 Power Issues

4.3.1 Power-up and Reset

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 which 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 (WIRE) spacecraft was lost when the FPGA produced unexpected outputs during power-up. The unexpected output resulted in the system reset process not completing, which lead to the early firing of a pyrotechnic device and ultimately to the failure of the mission [14].

4.3.2 Signal and Power Integrity Errors

Single ended signaling is often used on aircraft to reduce the weight of the wiring. In single-ended signaling, inputs and outputs (I/O) share a common power and ground connection. If all of the I/O connected to the common power supply or ground change state simultaneously,

a large spike in current will occur. Any parasitic inductances in the power supply and ground distribution network will have voltages induced across them which are proportional to the derivative of the current. These induced voltages are known as supply/ground bounce and can be large enough to lead to erroneous circuit operation.

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 (GND) and supply voltage (VDD) pins to the signal pin. Signal pins farther away from a GND or VDD pin are more susceptible to noise. This problem is exacerbated when a large number of I/O in the region switch simultaneously.

5 Conclusions

Technology has improved to the point that system designers have the ability to trade-off implementing complex functions in either hardware or software. However in the design tool world there are clear distinctions between software and hardware tools. One of the major concerns in any hardware design is assuring that the hardware correctly implements the HDL description. As the synthesis and place and route process proceeds the architecture used to implement any given HDL description can be changed to optimize the design for area, power, or timing. The synthesis tool views all of the implementations as logically equivalent, but they may not be equivalent in the eyes of the designer. Formal tools and specifically equivalence checking approach seem to be an excellent method to guarantee that the designer's intent has been translated to the physical hardware. While both verification of models and hardware synthesis have been successfully applied in industrial practice, there are several caveats in practice when physical components come in play. The issues are not due to the incorrectness of neither formal analyses nor errors in the synthesizers, but the inadequacy of the analyzed models and the not-so-simple internal conditions and related synthesizers' construction.

Formal methods must never give us a false sense of confidence. Despite the best design and verification efforts the hardware may still produce unexpected results. These errors can be due to noise, supply bounce, timing issues, or even cosmic radiation. It should be noted that specialized design tool suites to address all of the above error conditions exist. The tradeoffs between the costs and benefits of using these tools must be investigated for each design. Despite all of the design tools available, the most important component of any safety-critical design is an experienced designer with the experience and ability to differentiate between what issues are critical and what issues are negligible. It should be also noted that a rigorous process and safety culture promoted by appropriate guidance in regulated industries (e.g. FAA in aviation, FDA in the medical domain) is an integral element to improve safety.

Acknowledgements. The presented work was supported in part by the Aviation Airworthiness Center of Excellence under contract DTFACT-07-C-00010 sponsored by the FAA. Findings contained herein are not necessarily those of the FAA.

References

1. Pampagnin, P., Menis, J.F., DO254-ED80 for High Performance and High Reliable Electronic Components, Internal Paper, Barco-Siles S.A., Peynier, France, 2007.
2. Bernardo, M., Cimatti, A. (Eds.), Formal Methods for Hardware Verification, Proc. SFM 2006, 6th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, Bertinoro, Italy, May 22-27, 2006, Lecture Notes in Computer Science, Vol. 3965, Springer-Verlag, 2006
3. Kern, C., Greenstreet, M.R., Formal Verification in Hardware Design: A Survey, ACM Trans. on Design Automation of Electronic Systems, Vol. 4, No. 2, pp. 123-193, 1999.
4. Turner, K.J., He, J., Formally-based Design Evaluation, Proc. CHARME 2001, 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
5. Aljer, A., 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
6. Nehme, C., Lundqvist, K., A Tool for Translating VHDL to Finite State Machines, Proc. DACS 2003, 22nd Digital Avionics Systems Conference, October 12-16, 2003, Vol.1, pp. 3.B.6-1-7
7. Dajani-Brown, S., Cofer, 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
8. Hilton, A.J., High-Integrity Hardware-Software Codesign, Ph.D. Thesis, The Open University, April 2004
9. DO-254, Design Assurance Guidance for Airborne Electronic Hardware, RTCA Inc., Washington, DC, April 19, 2000
10. Dellacherie, S., Burgaud, L., di Crescenzo, P., imPROVE–HDL: A DO-254 Formal Property Checker Used for Design and Verification of Avionics Protocol Controllers, Proc. DACS 2003, 22nd Digital Avionics Systems Conference, Indianapolis, Ind., October 12-16, 2003, Vol. 1, pp. 1.A.1-1.1-8
11. Karlsson, K., 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
12. Henson, J., Equivalence Checking for FPGA Design, White Paper, Mentor Graphics Corp., Wilsonville, Ore., May 2007
13. IEEE Std 1076-2002, Standard VHDL Language Reference Manual, The Institute of Electrical and Electronics Engineers, New York, 2002
14. Bridgford, B., Carmichael, C., Tseng, C.W., Single-Event Upset Mitigation Selection Guide, Application Note XAPP987, Xilinx Inc., San Jose, Calif., March 2008