Proceedings Companion

Seventh European Dependable Computing Conference
EDCC-7

7-9 May 2008

Kaunas, Lithuania

Sponsored by:

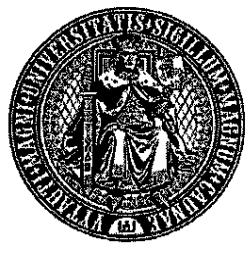
SEE Working Group "Dependable Computing", France
GI/ITG/GMA TC on Dependability and Fault Tolerance, Germany
AICA Working Group "Dependability of Computer Systems", Italy

In cooperation with:

IFIP Working Group 10.4 "Dependable Computing and Fault Tolerance"
EWICS TC7 on Safety, Reliability and Security
IEEE Computer Society TC on Dependable Computing and Fault Tolerance
Universidad Politecnica de Madrid, Spain
ReSIST, a European Network of Excellence

Hosted by:

Vytautas Magnus University, Kaunas, Lithuania



Vytautas Magnus University

Safety Concerns for Tool Use in the Design of Complex Electronic Hardware

Andrew J. Kornecki
Dept. of Computer & Software Eng.
Embry-Riddle Aeronautical Univ.
Daytona Beach, FL 32114
kornecka@erau.edu

Janusz Zalewski
Dept. of Computer Science
Florida Gulf Coast University
Ft. Myers, FL 33965
zalewski@fgcu.edu

Brian Butka
Dept. of Electrical & Systems Eng.
Embry-Riddle Aeronautical Univ.
Daytona Beach, FL 32114
butkab@erau.edu

Abstract

Modern electronic devices increasingly use dedicated hardware to process the growing amounts of data to control their operation. These complex programmable electronic devices are developed by writing code in a hardware description language used to create logic designs. Most of these devices, such as FPGA, can be configured to implement a particular design by downloading a sequence of bits. In that sense, a circuit implemented on an FPGA is literally software. However, treating circuits as "hardware" poses problems in system development and certification. The objective of this paper is to discuss the need for appropriate treatment of software processes in the development of complex electronic hardware, in a view of certification. We focus on software tools for hardware development in avionics and aerospace systems, and describe a method and procedures to evaluate a tool with respect to real-time constraints, for the purpose of qualification.

1. Introduction

Modern electronic systems use not only increasing numbers of microcomputers and microprocessors but also dedicated hardware to process the growing amounts of data needed to control systems operation and monitor their status. These complex programmable electronic devices are not only programmed using conventional languages but also are developed by writing code in a hardware description language used to create logic designs. 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 primary devices in this category include equipment based on PLD (Programmable Logic Devices), FPGA (Field Programmable Gate Arrays), ASIC (Application Specific Integrated Circuits), and similar circuits used as components of programmable electronic hardware. Often the circuit includes dedicated processors whose Intellectual Property (IP) is made into the final product silicon.

It is interesting to note that creation of the complex circuits is currently not considered a software activity and it is left to hardware specialists. However, the hardware/system description languages, such as VHDL, Verilog, or System C are basically computer languages with their own syntax and semantics. 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. It is the contention of the authors that the development of systems containing both software and hardware components needs to be done in a unified manner.

2. Concerns for Certification of Complex Electronic Hardware

The critical issue in the development of dependable safety-critical systems is the close relation between what used to be termed under separate categories: software vs. hardware. Software application designers focusing on development of programs to run on microprocessors, often overlook programmable logic devices and most popular Field Programmable Logic Arrays (FPGA). FPGA is a prefabricated integrated circuit 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 literally software. However, circuit designers are still known as hardware specialists, and the algorithms ported to circuits are still known as hardware algorithms. Treating circuits as "hardware" poses problems in computing system development, especially for embedded systems [1].

Of special concern is the fact that using both software and hardware in creation of dependable, often safety-critical systems requires meeting certain government regulations and engineering standards. For example, in the development of airborne systems installed on civilian aircraft, the software aspects of certification are guided by RTCA DO-178B "Software Considerations in Airborne Systems and Equipment Certification", which defines the processes and artifacts to meet the approval objectives.

On the other hand, RTCA DO-254, "Design Assurance Guidance for Airborne Electronic Hardware", provides details on the processes that must be followed in assessment of hardware components. In particular, the document provides certification information on project conception,

planning, design, implementation, testing, and verification. Both RTCA documents address the issue of qualification of the tools used for creation of an airborne system in a slightly different way. It is thus conceivable that a system of specific level of safety assurance (defined by the categories from A to D, i.e., from the most to the least critical) will receive different scrutiny depending whether it is implemented in software or in hardware.

Certification issues in aerospace applications are also becoming increasingly evident and important [2]. For example, decision software installed in unmanned autonomous systems (UAS) needs to be both reliable and safe. However, trusting decisions made by autonomous control software may require new methods and processes to guarantee safety. The difficulty lies in determining how these intelligent systems will operate in a dynamic environment and with little or no human oversight. UAS autonomous control requires significant technology advancement, since it does replace years of training for human operators and their decision-making abilities. Neglecting autonomous control certification research today will dramatically increase cost for future users [3].

The concern is that a designer can freely choose to implement a task in a mix of hardware and software, but the FAA regulations and the software tools struggle handling this type of a design. The objective of this paper is to discuss the need for proper treatment of software processes in the development of complex electronic hardware, in a view of the need for unification of software and hardware development for of certification. In the previous work [4], we applied the criteria for assessment of software development tools for safety-critical real-time systems. In this paper, we focus on software tools for development of complex electronic hardware, and describe a method and procedures to evaluate a tool with respect to real-time constraints for the purpose of qualification.

3. Typical Development Process for an FPGS Based System

A typical design, verification and validation flow for an FPGA based avionic system is shown in Figure 1. DO-254B requires that all verification of Level A and B functions should be independent. For level C and lower, independence is needed only at the design hierarchy where it is verified against the requirements. It is not the intent of independence to require someone other than the designer to execute the tests once they are evaluated or developed with independence. However, the results may still need to be reviewed independently to confirm proper procedures were followed and that the results verify that the requirements were met. Verification process objectives may be satisfied through a combination of reviews, analyses, and the development and execution of tests. The verification activities, such as test, simulation, prototyping, analyses and reviews should demonstrate compliance with the requirements. Since the complex electronic devices can be considered on various hierarchy levels (transistors, gates, circuits, components), it is not intended that requirements should be verified at every hierarchical level.

Many companies choose to generate internal verification and validation testbenches to provide rapid feedback and better assure design correctness. Figure 2 shows the design flow for a company utilizing internal verification techniques. The process begins with using the system requirements document to generate the hardware design specification. The design group will then implement the hardware design specification by writing Register Transfer Language (RTL) source code. RTL source code is typically written in using the Verilog or VHDL language. A separate group will use the system requirements document to generate the test plan and create a testbench. The test plan will typically consist of one or more of the following:

- Directed Tests generated to directly verify each requirement in the hardware systems document. DO-254 explicitly requires these tests.
- Constrained Random Tests consist of test vectors generated by defining a space of all legal system inputs and generating random sequences of legal inputs. This method often finds bugs that no amount of directed test does.
- Formal Proofs of correctness where the system is proven to generate the correct output for all legal inputs using mathematical and logical theorems.
- Error Condition Tests consist of applying illegal inputs to the system and verifying that system can recover from unexpected/illegal inputs in a safe manner.

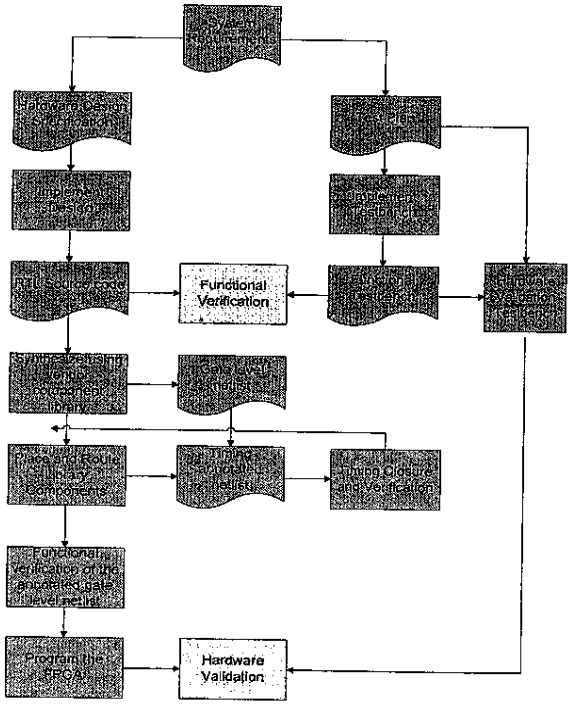


Figure 1. Typical design, verification and validation flow for an FPGA based avionic system.

The functional verification testbench in this technique typically requires many hours of simulation time to run. A typical procedure is that every evening all of the designs are checked into the design repository and functional verification is run on the most recent iteration of the design. Next the design group gets a report including every test that failed and the exact failure that occurred. When the functional verification is first being run, thousands of errors are often generated. The two groups must work together to identify whether the problem is in the design or in the testbench. Although tedious, this is often one of the most important tasks to be performed since it is common to find that the design and verification groups interpret the same specification in different ways. Clearing up errors in the system specification or its interpretation as early as possible can save many months of wasted design effort.

Once the design passes all of the functional tests it is synthesized into the hardware using the vendor's hardware library. The output of this step is called a gate level net which describes the design in terms of actual hardware available on the target device. For any given RTL source code there can be many possible implementations. The designer guides the tool in choosing the implementation by providing constraints on the design. The most common constraints are timing constraints which specify the acceptable delays on signal paths as well as register setup and hold constraints. Additional constraints can be area constraints where the amount of FPGA resources (or area) consumed are restricted, or such constraints as power dissipation.

The place and route tool implements the design in hardware and generates timing, area and power estimates for all signals. These estimates are then compared to the timing, area, and power constraints. If the current iteration does not meet the specification, the place and route tool attempts to meet the constraints by changing the physical location of the components and by changing the actual hardware implementation of the RTL source code. Although there are often multiple constraints involved, the most common problems are timing problems and this phase of the design is called timing closure. Timing closure of a large design will typically require several weeks of effort. It is possible for a timing problem to be so severe that the system architecture needs to be modified. In this case the hardware requirements will be modified, new RTL is generated, function verification performed again and then timing closure attempted using the new gate level netlist.

After the design meets timing closure the final gate level netlist is annotated with accurate timing information and functional verification is performed again using the best simulation model of the hardware. Assuming the final functional verification passes, the bitstream data for the FPGA is generated and downloaded. At this point the FPGA hardware is validated using laboratory equipment. The validation tests typically include all the test vectors used in the functional verification as well as vectors intended to check problems that cannot be easily seen in simulation.

The procedure presented above is consistent with the DO-254 guidelines that say: "as the complexity of the hardware design increases, it is advantageous to make use of computerized tools, such as simulation to verify requirements and implementation of the design." The issue is that it is a software program (written in an RTL) processed and analyzed by other software (tool) running on a general purpose workstation with conventional operating system (such as Windows) responsible for creation of hardware. The correctness and dependability of a software tool is critical to the effective and efficient creation of modern hardware.

4. Focus on Safety Issues in the Complex Hardware Development

The case studies are being developed based on the expressed concerns of the scientific and industrial communities with reference to complex electronic hardware (CEH) tools, such as those used for FPGA development. Largely these concerns are with relevance to development of safety-critical and real-time systems [4].

These case studies are geared towards qualifying the CEH tools. In an attempt to qualify the tools, the tools are going to be used with worst case scenarios along with least likely uses in order to test the bounds of the tools capability. The concept is that 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. This however shall be done in a design independent fashion. In order to facilitate the design independence, case studies are developed as small very focused experiments that are used to discover specific attributes of a tool. This method is preferred rather that elaborate designs, due to the fact that often broken links in tracing from design to implementation are due to a flaw in the design. The individual case studies specific focus is as follows:

1. Real timing – Determination of timing based on synthesis redesign. Also used to determine the applied safety margin to the reported timings.
2. Power constraints – Systematic way of determining if place and route functions are effective.
3. Undefined I/O status – Test to determine how the tool uses the I/O not defined by the user.
4. Simulation error – Behavioral simulation results vs. actual implemented behavior; to determine accuracy and reliability of the simulation portion of the tools.
5. Faulty hardware detection – Test to determine if a tool will attempt to implement design on faulty hardware.

Thorough literature research, surveys, and industry interviews, uncertainties and faults regarding the usage and/or operation of the tools has 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: if a user tries to implement something physically impossible, does the tool notify the user, alter the design to make it possible, attempt to implement the design, etc.? This leads to another topic in the scope of the case studies, does the tool have "awareness" of the hardware physical limits, or is this up to the user to oversee. 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 traces to safety constraints or timing constraints.

The majority of the tools come in a package that encompasses everything from coding or formal requirements/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 even self validation of formal requirements/design. This raises the question: if the tool does all the design/redesign and it verifies its own design will it ever really know if it is correct? This also begs the question of the independence of the verification that is required by DO-178B and DO-254.

For example, the purpose of Real-Time Constraints case study is to determine if a circuit meets the timing constraints that the tool displays in the design report and that margin of safety is retained. 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 minimums, the tool will redesign the circuit so that the delay is smaller. The subjects of interest are: where are these bounds, how close does the tool allow the design to get to the bounds, and is there a safety margin with actual delay and estimated delay?

The case study shall be implemented with an asynchronous ripple counter. An asynchronous ripple counter is a series of N flip-flops cascaded together so that the output of a flip-flop is clock input of the next one; this shall be done with five J-K flip-flops in toggle mode. The input to the ripple counter shall be an external pin that connected to a high frequency source. The final output of the counter shall be connected to an output pin. The expected output signal frequency shall 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.

Since experiments are designed to run on variety of platforms (i.e. combination of tool/board), the instruction set shall be written so that it is platform independent. For each platform it will be necessary to create different configuration files to utilize the development board I/O's. The design shall be synthesized and configured to each platform independently. Each platform will require that timing constraints are set for the ripple counters. It is acceptable that each platform timing constraints will be different. A timing report will be generated by the tool.

This and all the following experiments require a radio-frequency (RF) generator, function generator, oscilloscope capable of analyzing RF or RF specific measuring equipment, and a logic analyzer. The implementation shall be tested in two iterations. The first iteration shall be done with the system default timings. The second iteration shall be done with the tightest allowable timing constraints. The tight timing constrains require the tool to alter the design so that higher frequencies are obtainable.

The experiment requires manual sweep of the input pin from 0 Hz to the highest obtainable frequency in the GHz range (higher than the maximum reported gate frequency of the hardware). During the sweep both the input and output wave of the component shall be scoped for accuracy and phase differences. The input frequency shall 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 shall be recorded accurately for analysis.

5. Conclusion

If a system designer chooses to implement a design utilizing processors that are embedded inside significant hardware, the resulting system becomes a problem for safety-critical applications. Neither RTCA guidance DO178B (software) nor DO-254 (hardware) succeed in addressing the problems that are unique to a mixed system. Despite the design and verification efforts applying the appropriate hardware or software design tools, there is a significant concern that some errors can potentially slip through. This paper analyzes the related issues and describes a method and procedures to evaluate a tool with respect to real-time constraints, power analysis, and I/O issues for the purpose of qualification. One part of the solution might be that an anonymous database of known issues be compiled and that this database be used as a collective knowledge base for the use of software tools in design of complex electronic hardware for safety-critical systems. The case studies reported in this paper may be used to begin the system designer's knowledge base.

Acknowledgements

This work has been done under a contract supported by the Federal Aviation Administration DTFACT-07-C-00010. Contribution of MSE graduate student Jason Firanski to creation of the test cases is appreciated.

References

[1] Vahid F., It's Time to Stop Calling Circuits "Hardware", *Computer*, Vol. 40, No. 9, pp. 106-108, September 2007.

[2] Kornecki A. and Zalewski J., Software Development Tool Qualification from the DO-178B Certification Perspective, *CrossTalk – The Journal of Defense Software Engineering*, Vol. 19, No. 4, pp. 19-22, 2006.

[3] Jacklin S.A. 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*, Arlington, Virginia, Sept. 26-29, 2005, Paper No. AIAA 2005-6912.

[4] Kornecki A. and Zalewski J., Experimental Evaluation of Software Development Tools for Safety-Critical Real-Time Systems, *Innovations in Systems and Software Engineering – A NASA Journal*, Vol. 1, pp. 176-188, 2005.