Tool Qualification and Hardware Certification for Avionics and Aerospace Applications

Brian Butka
Electrical & Systems Engineering
Embry-Riddle Aeronautical University
Daytona Beach, FL 32114, USA
386-226-7752
butkab@erau.edu

Janusz Zalewski
Dept of Computer Science
Florida Gulf Coast University
Fort Myers, FL 33965, USA
239-590-7317
zalewski@fgcu.edu

Andrew Kornecki
Computer & Software Engineering
Embry-Riddle Aeronautical University
Daytona Beach, FL 332114, USA
386-226-6888
kornecka@erau.edu

ABSTRACT
Modern aircraft and space vehicles increasingly use dedicated hardware to process the growing amounts of data to control the flight. 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 discuss the need for methods and procedures to evaluate tools with respect to such issues as real-time constraints, power analysis, and I/O analysis.

Categories and Subject Descriptors
D.2.2 [**Software Engineering**]: Design Tools and Techniques – *computer aided software engineering (CASE).*

General Terms
Measurement, Design, Experimentation, Languages, Verification

Keywords
Software Tools, Programmable Logic, Testing

1. CONCERNS FOR CERTIFICATION OF COMPLEX ELECTRONIC HARDWARE
Modern aircraft and space vehicles use not only increasing numbers of microcomputers and microprocessors but also dedicated hardware to process the growing amounts of data needed to control the flight and related systems, and monitor their status. These complex programmable electronic devices are not only programmed using conventional programming 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."

Of special concern is the fact that using both software and hardware in creation of dependable, often safety-critical systems requires to meet certain government regulations and engineering standards. For example, in the development of airborne systems installed on civilian aircraft, the software aspects of their 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.

Certification issues in aerospace applications are also becoming increasingly evident and important [1]. 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, with little oversight. New paradigms are needed to assure safety. Intelligent control adds a whole new dimension to certification issues for flight control technologies including the most rigorous testing. UAS autonomous control requires significant technology advancement. Neglecting autonomous control certification research today will dramatically increase tomorrow's cost for future users [2, 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 the purposes of certification. In the previous work [4], we developed and applied the criteria for assessment of software development tools for safety-critical real-time systems. In this paper, we focus on selected issues in tool use related to timing constraints, power analysis, and I/O.

1st Workshop on Mixed Criticality, April 16, 2009, San Francisco, Calif.

2. FOCUS ON SOFTWARE TOOLS FOR COMPLEX ELECTRONIC HARDWARE

The case studies are being developed based on the expressed concerns of the scientific and industrial communities regarding 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]. They are geared towards qualifying the CEH tools. The individual case studies specific focus is on: power constraints, undefined I/O status, timing, simulation errors, and faulty hardware detection. Here, we only briefly mention the first two case studies.

2.1 Power Analysis

The purpose of this case study is to determine if a CEH tool successfully routes connections within an FPGA, so that electromagnetic fields and maximum current draws do not affect the logic or output voltage levels. The worst case scenario with reference to power consumption and electromagnetic fields is when all I/O pins of an FPGA are sinking at their max. current load and all of the components within the FPGA are active.

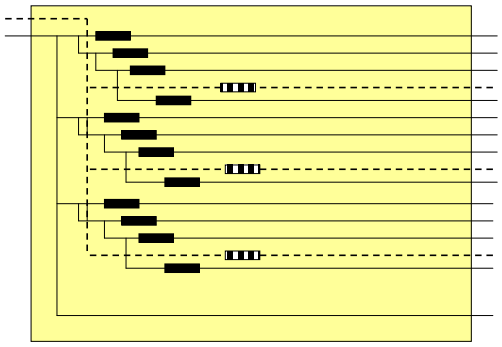

Figure 1. Power and I/O Analysis Test Cases.

In Figure 1 the solid lines represent variable frequency lines and the dotted lines represent the known signal paths. The boxes are ripple counters. All variable frequency ripple counters shall have the timing set so that they are capable of operation at their highest allowable frequency. This forces the tool to expand the ripple counter design using more gates. This constraint is both tool and hardware dependent. The known logic lines shall have a fixed known frequency divisor.

While running the sweep, all nodes connected to current-sinking devices need to be monitored. The signal shall be monitored at one of the pins for correct frequency, voltage levels, and rise/fall times. The path of known logic shall be monitored for correct logical readings, voltage levels, and noise due to interference.

2.2 I/O Analysis

Of several issues in I/O analysis, we focus here on Simultaneous Switching Noise (SSN). It is not uncommon for FPGAs to implement data buses that are 32 or even 64 bits wide. SSN occurs if a large number or all of the signals switch from 0 to 1 or 1 to 0 simultaneously. Switching many outputs stresses the internal power networks and can lead to unexpected failures.

To test SSN performance, the FPGA is configured to switch the maximum number of outputs. The SSN behavior is examined with several different loads, because LVTTL loads can be modeled in several ways. First, the loads are rated at an output current of 24 mA, so resistive loads are used to draw this much current from the I/O. The circuit resistor is terminated to the mid-rail since it is not known whether switching from 0-1 or 1-0 is the worst case. This allows both transitions to be easily investigated.

Table 1. Results of experiments to detect simultaneous switching noise.

Input Voltage	Output from spec	Output Resistive Load	Output Capacitive Load
0.0	0	0	0
0.5	0	0	
1.0		0	0, occasional glitch
1.5		1	continuous glitching
2.0	1	1	1, occasional glitch
2.5	1	1	1, occasional glitch
2.5	1	1	1
3.0	1	1	1
3.3	1	1	1

Results of experiments are presented in Table 1. An input was located near many simultaneously switching outputs. The input was swept from 0 to VDD (3.3V). The output was compared to the spec requirements. The resistive load case passes spec and the capacitive load case fails. The failing conditions are underlined.

3. 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 nor DO-254 succeed in addressing the problems unique to a mixed system. Despite the design and verification efforts applying the appropriate design tools, there is a significant concern that some errors can potentially slip through. This paper analyzes the related issues and suggests procedures to evaluate a tool with respect to power analysis and I/O for the purpose of qualification.

4. ACKNOWLEDGMENTS

This work has been done under a contract from Federal Aviation Administration DTFACT-07-C-00010.

5. REFERENCES

[1] 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, April 2006.

[2] Crum V., Homan D. and Bortner R., Certification Challenges for Autonomous Flight Control Systems, *Proc. AIAA Guidance, Navigation, and Control Conference and Exhibit*, Provdence, R.I., August 16-19, 2004, Paper No. AIAA 2004-5257.

[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 and Exhibit*, 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.